REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-05-

0423

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 31, 2001	3. REPORT TYPE AND DATES COVERED Final Performance Report 14 Nov 00 – 31 February 05

4. TITLE AND SUBTITLE **TIME SENSITIVE CONTROL OF AEROSPACE OPERATIONS**	5. FUNDING NUMBERS **F49620-01-1-0008**
6. AUTHORS Abbas Zaidi Lee W. Wagenhals	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) George Mason University Center of Excellence in Command, Control, Communications, and Intelligence Mail Stop 4B5 Fairfax, VA 22030-4444	8. PERFORMING ORGANIZATION REPORT NUMBER GMU/C3I-235-R
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM (Dr. Neal D. Glassman) 801 N. Randolph Street, Arlington, VA 22203	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research is motivated by an effort to integrate temporal and spatial aspects of a large-scale discrete event system (DES) for dynamic control (i.e., planning and re-planning) purposes. The dynamic planning and re-planning problems require satisfaction of two types of constraints, namely temporal and spatial, and the challenge here is to integrate both these constraints in a single analytical formulation of the problem. In addition, the constraints may not be all quantitatively specified; therefore, there is a need to have provisions for both quantitative and qualitative constraint-handling in this formulation. The dynamic control of Discrete-Event Systems (DES) often requires revising a produced temporal model during and/or after system specification phase, e.g., the constraints or system/mission requirements may change during or before a plan's execution. This research resulted in extensions to the existing point interval logic (PIL) that addresses these challenges. The extension allows for a larger class of temporal systems to be handled by incorporating an enhanced input lexicon, representation of flexibility in temporal specifications, an improved verification and inference mechanism, and a suite of analysis tools. The PIL formalism has been shown to incorporate temporal and spatial information separately from each other. The present implementation offers a toolkit of graph-based algorithms for implementing inference and verification mechanisms, revision algorithms for identifying the type of change before making the revision to the plan, and the application of the logic for temporal and spatial knowledge representation and reasoning. A recent attempt on combining the graph-based revision and the inference mechanisms into an algorithm that can efficiently handle the dynamic change has yielded promising results. Applications of the logic include temporal planning and temporal analysis of Effects-based Operations (EBO) and the results of this research have been transitioned to the Air Force Research Laboratory IF Division at Rome, NY.

14. SUBJECT TERMS Point-Interval Logic Temporal Logic, Spatial Logic, Dynamic Planning and Control of Discrete Event Systems, Effects Based Operations	15. NUMBER OF PAGES 107
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

MAY 2005 GMU/C3I – 135-R

CENTER OF EXCELLENCE IN C3I
GEORGE MASON UNIVERSITY
Fairfax, VA 22030

Performance Report

TIME SENSITIVE CONTROL OF AEROSPACE OPERATIONS

FINAL TECHNICAL REPORT
Contract No. F49620-01-1-0008
For the period
November 14, 2000 to February 14, 2005

Submitted to:

Dr. Neal D. Glassman (703) 696-9548
AFOSR/NM
801 North Randolph Street
Arlington, VA 22203-1977

Submitted by:

Dr. Lee W. Wagenhals
Principle Investigator
4400 University Dr.
Fairfax, VA 22030
May 31, 2005

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

George Mason University was awarded a grant to address significant theoretical challenges to support the dynamic planning and execution of expeditionary aerospace forces. In particular, this requires special attention to temporal and spatial issues. The need to carry out a whole spectrum of operations with very short staging times and far away from CONUS bases means that resources must be managed both spatially and temporally. The integration of temporal and spatial logic is the new direction of research that this effort addressed. This is the final technical report of those activities. It is organized into three sections. Section 1 includes this introduction and provides a description of the tasking under the grant. Section 2 provides a summary of the motivation, approach, and results. Section 3 provides detailed descriptions of the research effort including the formal definitions of the temporal logic that was developed, the adaptation of that temporal logic to a spatial logic, and the integration of the two logics. A description of the software applications that were developed is also provided.

1.1 Tasking

The following task statements describe the research under this contract:

Task 1: Extend the capability of the Temporal Programmer, TEMPER, to address the general case for systems with multiple time lines with multiple futures (MTMF)

A multiple time lines with multiple futures (MTMF) system is characterized either by a single set of events with associated multiple time lines each yielding a different future (type 1) or by multiple set of events with different single/multiple time sequences each representing a total world-history (type 2). In addition to enhanced capabilities in terms of the type of temporal systems, the tool will have a web-based interface to facilitate collaborative planning and re-planning in the time domain.

Task 2: Integrate Temporal and Spatial Paradigms

The dynamic planning and re-planning problems require satisfaction of two types of constraints, namely temporal and spatial. The time-sensitive aspect requires a planner to sequence time intervals (points) associated with mission activities without violating any of the system specifications, given a priori. The spatial aspect, on the other hand, needs to look at the availability of the physical resources capable of handling the required task list. The challenge here is to integrate both these constraints in a single analytical formulation of the problem and devise an engine that can

(i) model the two aspects using a single integrated formalism,

(ii) validate the feasibility of all types of constraints, and

(iii) help generate a feasible plan, given additional mission requirements.

Task 3: Generate Time Phased Plans using Occurrence Graph analysis of a Colored Petri net model consisting of a physical layer, a temporal layer, and a collaboration layer.

The collaborative generation of feasible time phased plans, either at the initial planning stage or during dynamic battle control, which is based on changes in the environment, changes in tasks to be performed or changes in resources available, can be formulated as a three layer problem using a physical layer, a temporal layer, and a collaboration layer. There is a set of tasks, which must be performed to accomplish a goal (e.g., execute a mission or complete an operation) and a set of resources, which can be used to execute the tasks. At the physical layer, the relationship between tasks and resources capable of executing the tasks is specified along with the resource capabilities (time to execute given tasks, multitasking capability, etc.) At the temporal layer, the sequencing constraints between tasks, required task completion times, etc., are modeled using a temporal logic model such as the Point-Interval temporal Logic (PITL) of Zaidi. At the collaboration layer, the merging of temporal requirements and physical resource constraints to determine the feasible time phased plan is accomplished so that both the temporal constraints and the resource constraints are satisfied. In this task, the collaboration will be modeled in the form of Colored Petri nets and analyzed by occurrence graph analysis of the nets.

2. SUMMARY OF RESEARCH

2.1 Motivation

The tasks in this research are motivated by an effort to integrate temporal and spatial aspects of a large-scale discrete event system (DES) for dynamic control (i.e., planning and re-planning) purposes. As stated under Task 2, the dynamic planning and re-planning problems require satisfaction of two types of constraints, namely temporal and spatial, and the challenge here is to integrate both these constraints in a single analytical formulation of the problem. In addition, the constraints may not be all quantitatively specified; therefore, there is a further need to have provisions for both quantitative and qualitative constraint-handling in this formulation. The research is further challenged by the fact that the dynamic control of Discrete-Event Systems (DES) often requires revising a produced temporal model during and/or after system specification phase, e.g., the constraints or system/mission requirements may change during or before a plan's execution.

2.2 Accomplishments

The results of the research under the contract have successfully solved some of the issues and posed both challenges and promise in resolving others. The research effort has resulted in the formulation of point-interval logic and an implementation of its inference engine in the form of a software application. The application implements the *Pointisable* temporal algebra [Vilain et al., 1990] with provisions for metric temporal information for both points and intervals. The logic is shown to model and reason about temporal and spatial information. Applications of the logic include temporal planning and temporal analysis of Effects-based Operations (EBO) and the results of this research have been transitioned to the Air Force Research Laboratory IF Division at Rome, NY.

2.3 Previous Research

The point-interval formalism was first reported in [Zaidi, 1999] as an extension of Hamblin's time primitives (1972), and Allen's Interval logic [Allen, 1983]. (Similar extensions to Allen's ontology have also been reported by several other researchers [Meiri, 1991; Drakengren and Jonsson, 1997; Ma and Knight, 2001].) This extension allows the provision of points (i.e., intervals with zero lengths) in Allen's ontology. The inclusion of point as a primitive entity was done due to the reason that a system's temporal aspects might be represented in terms of properties that hold for certain time intervals, processes taking some time to complete, but can also be represented as events/occurrences requiring virtually no time to take place, e.g. instantaneous events. In order to model a system's temporal aspects, one, therefore, needs to have a formalism capable of handling both interval and point descriptions of various components of the system. A similar argument can be made for modeling spatial aspects of a system. This formalism extends the axiomatic system of interval logic by adding new axioms to it. A Petri net [Peterson, 1981; Reisig, 1991] model is shown to represent this axiomatic system by transforming the system's specifications (i.e. qualitative temporal/spatial relations between system entities) given by statements of point-interval logic into Petri net structures. The Petri net structure, with some of its analytical tools, was subsequently renamed Point Graph (PG). The temporal version of the point-interval formalism is called point-interval temporal logic (PITL). A temporal inference engine based on this Point Graph representation infers new temporal relations among system intervals, identifies temporal ambiguities and errors (if present) in the system's specifications, and finally identifies the intervals of interest defined by the user. The inference engine is also implemented in a tool, called TEMPER, for a subclass of PITL, called Single Timeline Single Future (STSF) systems [Zaidi, 1999].

Research described in [Zaidi and Levis, 2001] extends the point-interval approach one step further by adding provisions for 'dates/clock' times and time 'distances' for points and intervals. This extension allows the assignment of actual lengths to intervals, time distances between points, and time stamps to points representing the actual time of occurrences, whenever such information is available. The paper, therefore, addresses the issue of combining qualitative and quantitative information into a single formalism. The paper also presents the modified tool, TEMPER-II, which implements the inference engine for STSF systems with both qualitative and quantitative temporal information. The temporal inference engine of TEMPER-II infers temporal relations among system's intervals, identifies the windows of interest to the user, calculates lengths of intervals and windows, and infers actual time of occurrence of events.

A further extension [Zaidi, 2000] in the PG formalism incorporates the imprecise quantitative information to be represented and reasoned about. The new approach is capable of handling statements of the form "The length of an interval A is between 10 and 20 time units", and "An event B occurs somewhere between 0900 and 1100 hours." The temporal systems with this type of quantitative imprecision are termed as Multiple Timelines Multiple Future (MTMF), Type I, Systems. [Zaidi, 2000] The imprecision in quantitative temporal information is handled with the help of Hierarchical Point Graphs (HPG). The paper [Zaidi, 2000] discusses the progress made in modeling MTMF-I systems with HPGs, and identifies certain key strengths of HPGs in modeling temporal aspects of DESs over other contemporary approaches, e.g. temporal constraint networks [Dechter et al. 1991].

2.4 Current Results

The dynamic control of DESs often requires revising a produced model during and/or after system specification phase, e.g., the constraints or system/mission requirements may change during or before a plan's execution. The implementation of the temporal logic developed in earlier research called TEMPER does not support revision (add, modify, delete); even a minor change could not be made unless one re-edited the statements and restarted the TEMPER's engine from scratch. It requires the process of modeling to be re-initiated with the new set of unchanged and revised inputs. This limitation makes it infeasible for use in planning real time systems and other dynamic application areas, especially with large temporal inputs. This weakness in the Point Graph formalism was overcome in [Rauf and Zaidi, 2002]. The changes in the temporal specifications are classified into 'local,' 'regional,' and 'global' based on the impact of the change on the original set. [Ma, 1999] The approach in [Zaidi and Rauf, 2002] uses a multi-layered PG structure to keep the input specifications in the lowest layer of a PG. The qualitative and quantitative information available to TEMPER is processed and kept at a higher layer in the PG. The inference engine works at the higher layer to answer queries and infer new temporal relations; however, a change in the inputs is processed at the lowest layer and its 'effects' are propagated upwards. The affected parts of the PG determine the kind of change required of TEMPER to accommodate.

Zaidi and Wagenhals (2004) consolidated the results of the previous work on the point-interval logic and its application to model and plan time-sensitive aspects of a mission by further extending the approach. The extension allows for a larger class of temporal systems to be handled by incorporating an enhanced input lexicon, representation of flexibility in temporal specifications, an improved verification and inference mechanism, and a suite of analysis tools. The paper [Zaidi and Wagenhals, 2004] also presents a planning application of the formalism that identifies the *critical* activities, time slacks for the non-critical activities, and offers a graph based tool for 'what-if' analysis of the plan.

The underlying point-interval formalism presented in [Zaidi, 1999, 2000; Zaidi and Levis, 2001; Rauf and Zaidi, 2002] is shown, in [Hussain and Zaidi, 2002], to model the spatial information in a 1-dimensional case, with temporal components of the lexicon replaced by their spatial counterparts. The logic, however, is extended to incorporate 'orientation' aspects in spatial reasoning. The PG formalism is shown to handle the orientation aspect with the help of a new set of axioms in the logic. The spatial version of TEMPER is called SPINE (Spatial Inference Engine.) Another paper [Zaidi, Rizvi and Hussain, 2003] builds upon the logic proposed for reasoning over spatial relations in one dimension, forming spatial relations for objects in space using a set of basic relations. A point, a vertical line, a horizontal line, and a rectangle, are taken as primitive spatial objects in this approach. The spatial specifications are input by identifying the objects, their shape characteristics, and known spatial relationships among objects using a spatial lexicon. The representation of an object in space is determined by the projections of the object on both x and y axes. The graph-based representation of PGs is used to model the x and y projections of the objects in space. The graph-based spatial inference engine (SPINE) identifies spatial ambiguities and errors (if present) in the system's specifications, infers new spatial relations among objects' x and y projections, exactly like its temporal counterpart with an added capability of orientation. A second set of axioms is added in SPINE to enhance its inference capabilities for 2-dimensional case. The new set of axioms takes the SPINE inferences about the objects' x and y projections and infers their qualitative spatial relationships in space.

3. TECHNICAL COMPONENTS

This section presents the point-interval formalism, called Point Interval Logic (PIL), and the implementation of its inference mechanism with the help of a graph-based approach, called Point Graphs (PG). The PIL models point and interval descriptions of events on a real number line. The real number line may represent a time line and/or a coordinate axis where lengths of actual physical objects are mapped as intervals. The points and intervals, therefore, may represent time stamps and time delays, or lengths and distances between spatial objects. This gives rise to both temporal and spatial treatment of these points/intervals in the logic. The temporal and spatial application of the logic is presented in the following sub-sections.

3.1 Point-Interval Logic (PIL)

Lexicon

The lexicon of the Point Interval Logic (PIL) consists of the following primitive symbols:

Points (Event): A point X is represented as [pX, pX] or simply [pX].

Intervals: An interval X is represented as [sX, eX], where 'sX' and 'eX' are the two end points of the interval, denoting the 'start' and 'end' of the interval, s.t. $sX < eX$. (In the sequel, the term interval is used to refer to both intervals and points, if not explicitly stated otherwise.)

Point Relations: These are the relations that can exist between two points. The set of relations R_P is given as: $R_P = \{<, =\}$

Interval Relations: These are the *atomic* relations that can exist between two intervals. The set of relations R_I is given as: $R_I = \{<, m, o, s, d, f, =\}$

Point-Interval Relations: These are the *atomic* relations that can exist between a point and an interval. The set of relations R_π is given as: $R_\pi = \{<, s, d, f\}$

Functions: The following two functions are used to represent quantitative information associated with intervals.

Interval length function that assigns a non-zero positive number to a system interval, e.g.,

$$\text{Length } X = d, \text{ where } X = [sX, eX], d \in \Re^+$$

The stamp function assigns a number to a system point, e.g., Stamp p1 = t, $t \in \Re$

Proposition 3.1.1

The PIL relations in sets R_P, R_I, and R_π are mutually exclusive and exhaustive, i.e.

(a) if 'X Ri Y', Ri is a PIL relation, then there does not exist another PIL relation Rj such that 'X Rj Y' also holds true;

(b) for any two intervals (points) X and Y there must exist an PIL relation Ri such that either 'X Ri Y' or 'Y Ri X' holds true (with the exception of '=' relation where 'X = Y' is equivalent to 'Y = X'.)

Note: The second part of the proposition only holds for a 'complete' system of PIL statements.

Syntactic and Semantic Structure

The syntactic and semantic structure of atomic relations in PIL is shown in Table 3.1.1 The table outlines three possible cases (i.e., interval-interval, point-interval, and point-point) and the corresponding semantically relevant relations that can exist between points and/or intervals, represented by generic symbols X and Y. A qualitative relation between two intervals can be described with the help of algebraic inequalities, also shown in Table 3.1.1, among points representing the start and end of these intervals. The readers are cautioned on the dual use of some of these symbols for representing both algebraic and PIL relations. The context of their use makes the distinction very clear and the different uses of the same symbol in two different contexts should not be confused with each other.

A system of PIL statements is given by a conjunction of statements each describing a PIL relation between a *unique* pair of intervals/points. Example 3.1.1 presents two syntactically correct systems of PIL statements.

Example 3.1.1

Δ1:	event1 s Process1	Δ2:	X o Y
	Stamp event1 = 1000		Length[sX, sY] = 10
	Length Process1 = 10		Length[sY, eX] = 8
	event2 f Process1		Z o Y
	event2 s Process2		Length[sZ, sY] = 5
	Length Process2 = 20		Length[sY, eZ] = 8

Two points, p1 and p2, on a real number line are related to each other by one of the following three algebraic relations: '<' (less/greater than), '=' (equal to), and '≤' (less/greater than or equal to). A relation Ri between two intervals X and Y, denoted as 'X Ri Y' can, therefore, be represented as a 4-symbol string made of elements from the alphabet {<, =, >, ≤, ≥, ?}, where the first (left-most) symbol represents the algebraic relation between sX and sY, second symbol between sX and eY, third symbol represents relation between eX and sY, and fourth between eX and eY. The '?' is added to incorporate incomplete information. Table 3.1.2 shows this string representation for each of the atomic PIL relations.

The provision of '≤' (and '≥') relation between two points in the string representation of Table 3.1.2 results in the definition of *compound* relations between points and intervals.

Definition 3.1.1: Compound PIL Relation

A compound PIL relation between a pair of intervals (or between an interval and a point) is defined to be a disjunction of two or more atomic PIL relations between the two intervals, i.e., (X <m Y) = (X < Y) ∨ (X m Y), where X and Y are intervals and/or points.

The approach presented here, however, does not allow all possible disjunctive combinations of PIL relations between intervals and points. The only allowable disjunctive combinations of PIL relations that can be used to construct compound relations are given in Tables 3.1.3-3.1.5. The definition of the allowable relations between intervals is done by the use of symbols from the alphabet {<, =, >, ≤, ≥, ?} in constructing the string representation of a relation. For the sake of brevity, Table 3.1.3 does not show the remaining inverse PIL relations (Definition 3.1.2) and their corresponding string representations.

Definition 3.1.2: Inverse Relation

(a) Let Ri be an atomic PIL relation. The inverse of Ri, denoted by Ri^{-1}, between two intervals X and Y, represented as X Ri^{-1} Y, is defined to be equivalent to Y Ri X;
The inverse of an inverse results in the atomic PIL relation, i.e., $(Ri^{-1})^{-1} = Ri$.

(b) Let ρ be a compound PIL relation. The inverse of ρ, denoted as ρ^{-1}, is obtained by inverting all the constituent atomic relations in ρ, i.e., $(mof^{-1})^{-1}$ is equal to $m^{-1}o^{-1}f$.

Axiomatic System

The inference mechanism of PIL uses the analytical representation of PIL statements presented in Tables 3.1.2 - 3.1.5 and the following Axioms to infer unknown relations among system intervals. The axioms have been generated by an exhaustive enumeration of all possibilities involving points and/or intervals.

A. Point Axioms

Let p1, p2, and p3 be points defined on a real number line.

(A.1) $(p1 < p2) \wedge (p3 < p1) \rightarrow (p3 < p2)$

(A.2) $(p1 < p2) \wedge (p3 = p1) \rightarrow (p3 < p2)$

(A.3) $(p1 < p2) \wedge (p3 \leq p1) \rightarrow (p3 < p2)$

(A.4) $(p1 < p2) \wedge (p2 = p3) \rightarrow (p1 < p3)$

(A.5) $(p1 < p2) \wedge (p2 \leq p3) \rightarrow (p1 < p3)$

(A.6) $(p1 = p2) \wedge (p3 = p1) \rightarrow (p3 = p2)$

(A.7) $(p1 = p2) \wedge (p3 \leq p1) \rightarrow (p3 \leq p2)$

(A.8) $(p1 = p2) \wedge (p2 \leq p3) \rightarrow (p1 \leq p3)$

(A.9) $(p1 \leq p2) \wedge (p3 \leq p1) \rightarrow (p3 \leq p2)$

(A.10) $(p1 \_ p2) \wedge (p3 \_ p1) \rightarrow (p3 ? p2)$

(The symbol '?' represents unknown relation. The symbol ' _' is used to denote remaining combinations of the relation, {<, =, >, ≤, ≥, ?}, not covered by axioms 1 – 10.)

B. Interval Axioms

Let X and Y be intervals; X = [sX, eX] and Y = [sY, eY]

(B.1) $(sX < sY) \rightarrow (sX < eY)$

(B.2) $(sX = sY) \rightarrow (sX < eY) \wedge (eX > sY)$

(B.3) $(sX \leq sY) \rightarrow (sX < eY)$

(B.4) $(sX > sY) \rightarrow (eX > sY)$

(B.5) $(sX \geq sY) \rightarrow (eX > sY)$

(B.6) $(sX > eY) \rightarrow (sX > sY) \wedge (eX > sY) \wedge (eX > eY)$

(B.7) $(sX = eY) \rightarrow (sX > sY) \wedge (eX > sY) \wedge (eX > eY)$

(B.8) $(sX \geq eY) \rightarrow (sX > sY) \wedge (eX > sY) \wedge (eX > eY)$

(B.9) $(eX < sY) \rightarrow (sX < sY) \wedge (sX < eY) \wedge (eX < eY)$

(B.10) $(eX = sY) \rightarrow (sX < sY) \wedge (sX < eY) \wedge (eX < eY)$

(B.11) $(eX \leq sY) \rightarrow (sX < sY) \wedge (sX < eY) \wedge (eX < eY)$

(B.12) $(eX < eY) \rightarrow (sX < eY)$

(B.13) $(eX = eY) \rightarrow (sX < eY) \wedge (eX > sY)$

(B.14) $(eX \leq eY) \rightarrow (sX < eY)$

(B.15) $(eX > eY) \rightarrow (eX > sY)$

(B.16) $(eX \geq eY) \rightarrow (eX > sY)$

C. Point-Interval Axioms

Let X be a point and Y an interval; X = [pX] and Y = [sY, eY]

(C.1) $(pX < sY) \rightarrow (pX < eY)$

(C.2) $(pX = sY) \rightarrow (pX < eY)$

(C.3) $(pX \leq sY) \rightarrow (pX < eY)$

(C.4) $(pX > eY) \rightarrow (pX > sY)$

(C.5) $(pX = eY) \rightarrow (pX > sY)$

(C.6) $(pX \geq eY) \rightarrow (pX > sY)$

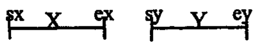
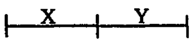
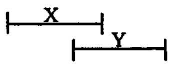
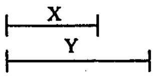
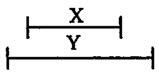
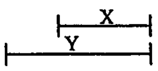
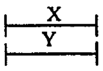
The inference mechanism of PIL constructs the analytical representation for the pairs of intervals with unknown relations with the help of the axioms. The resulting string representation of the relation(s) is pattern matched with the string representations of Tables 3.1.2-3.1.5 to infer possible relation(s) between the intervals. An inference engine for PIL, therefore, requires an exhaustive enumeration of the result through all feasible combinations of available statements, provided no knowledge of the system's correctness is available a priori. An inference engine that outputs the result as soon as it finds the first feasible set of inputs can only be applied to a known consistent system of PIL statements. This, in turn, requires a front-end verification mechanism for the PIL statements. Another point to note is that the axiomatic system presented in this section does not take into account the quantitative information that might be available to the system. Zaidi, in 1999, proposed a graph-based methodology, termed Point Graphs, to resolve these problems. A discussion on this methodology follows in the next section.

Table 3.1.1. Expressions in PIL and Their Semantics

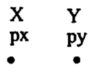
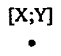
Qualitative Relations

CASE I— X and Y both intervals with non-zero lengths:

$X = [sX, eX]$, $Y = [sY, eY]$ with $sX < eX$ and $sY < eY$

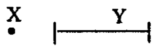
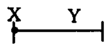
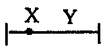
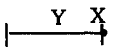
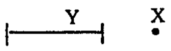
1.	$X < Y$	$eX < sY$	
2.	X m Y	$eX = sY$	
3.	X o Y	$sX < sY$; $sY < eX$; $eX < eY$	
4.	X s Y	$sX = sY$; $eX < eY$	
5.	X d Y	$sX > sY$; $eX < eY$	
6.	X f Y	$sY < sX$; $eY = eX$	
7.	$X = Y$	$sX = sY$; $eX = eY$	

CASE II—X and Y both points: $X = [pX]$ and $Y = [pY]$ with $sX = eX = pX$ and $sY = eY = pY$

1.	$X < Y$	$pX < pY$	
2.	$X = Y$	$pX = pY$	

CASE III— X is a point and Y is an interval:

$X = [pX]$ and $Y = [sY, eY]$ with $pX = sX = eX$ and $sY < eY$

1.	$X < Y$	$pX < sY$	
2.	X s Y	$pX = sY$	
3.	X d Y	$sY < pX < eY$	
4.	X f Y	$pX = eY$	
5.	$Y < X$	$eY < pX$	

Quantitative Relations

X, Y are points, Z is an interval and d is an integer

1. Stamp $X = d$	2. Length$[X, Y] = d$	3. Length $Z = d$

Table 3.1.2. Analytical Representation of PIL Relation

CASE I— X and Y both intervals with non-zero lengths:

• X = [sX, eX], Y = [sY, eY] with sX < eX and sY < eY

X Ri Y	**sX Vs. sY**	**sX Vs. eY**	**eX Vs. sY**	**eX Vs. eY**
X < Y	<	<	<	<
X m Y	<	<	=	<
X o Y	<	<	>	>
X s Y	=	<	>	<
X d Y	>	<	>	<
X f Y	>	<	>	=
X ≡ Y	=	<	>	=
X unknown Y	?	?	?	?

CASE II—X and Y both points:

X = [pX] and Y = [pY] with sX = eX = pX and sY = eY = pY

X Ri Y	**pX Vs. pY**
X < Y	<
X ≡ Y	=
X ? Y	?

CASE III— X is a point and Y is an interval:

X = [pX] and Y = [sY, eY] with pX = sX = eX and sY < eY

X Ri Y	**pX Vs. sY**	**pX Vs. eY**
X < Y	<	<
X s Y	=	<
X d Y	>	<
X f Y	>	=
Y < X	>	>
X ? Y	?	?

Table 3.1.3. Analytical Representation of Allowable PIL Relations Between Two Intervals

sX Vs. sY	sX Vs. eY	eX Vs. sY	eX Vs. eY	X Ri Y
<	<	<	<	<
<	<	=	<	m
<	<	>	<	o
<	<	>	≤	of^{-1}
<	<	>	?	$\mathrm{od}^{-1}\mathrm{f}^{-1}$
<	<	≤	<	<m
<	<	≥	<	mo
<	<	≥	≤	mof^{-1}
<	<	≥	?	$\mathrm{mod}^{-1}\mathrm{f}^{-1}$
<	<	?	<	<mo
<	<	?	≤	$<\mathrm{mof}^{-1}$
<	<	?	?	$<\mathrm{mod}^{-1}\mathrm{f}^{-1}$
=	<	>	<	s
=	<	>	=	=
=	<	>	≤	s=
=	<	>	?	$\mathrm{ss}^{-1}=$
>	<	>	<	d
>	<	>	=	f
>	<	>	≤	df
≤	<	>	<	os
≤	<	>	≤	$\mathrm{osf}^{-1}=$
≤	<	>	?	$\mathrm{oss}^{-1}\mathrm{d}^{-1}\mathrm{f}^{-1}=$
≤	<	≥	<	mos
≤	<	≥	≤	$\mathrm{mosf}^{-1}=$
≤	<	≥	?	$\mathrm{moss}^{-1}\mathrm{d}^{-1}\mathrm{f}^{-1}=$
≤	<	?	<	<mos
≤	<	?	≤	$<\mathrm{mosf}^{-1}=$
≤	<	?	?	$<\mathrm{moss}^{-1}\mathrm{d}^{-1}\mathrm{f}^{-1}=$
≥	<	>	<	sd

≥	<	>	=	f=
≥	<	>	≤	sdf=
?	<	>	<	osd
?	<	>	=	$ff^{-1}=$
?	<	>	≤	$osdff^{-1}=$
?	<	>	?	$oo^{-1}ss^{-1}dd^{-1}ff^{-1}=$
?	<	≥	<	mosd
?	<	≥	≤	$mosdff^{-1}=$
?	<	≥	?	$moo^{-1}ss^{-1}dd^{-1}ff^{-1}=$
?	<	?	<	<mosd
?	<	?	≤	$<mosdff^{-1}=$
?	<	?	?	$<moo^{-1}ss^{-1}dd^{-1}ff^{-1}=$
?	≤	≥	?	$mm^{-1}oo^{-1}ss^{-1}dd^{-1}ff^{-1}=$
?	≤	?	?	$<mm^{-1}oo^{-1}ss^{-1}dd^{-1}ff^{-1}=$
?	?	?	?	$<<^{-1}mm^{-1}oo^{-1}ss^{-1}dd^{-1}ff^{-1}=$

Table 3.1.4. Analytical Representation of Allowable PIL Relations Between Two Points

pX Vs. pY	**X Ri Y**
<	<
=	=
>	$<^{-1}$
≤	<=
≥	$=<^{-1}$
?	$<=<^{-1}$

Table 3.1.5. Analytical Representation of Allowable PIL Relations Between a Point and an Interval

pX Vs. sY	**pX Vs. eY**	**X Ri Y**
$<$	$<$	$<$
$=$	$<$	s
$>$	$<$	d
$>$	$=$	f
$>$	$>$	$<^{-1}$
$>$	$\leq$	df
$>$	$\geq$	$f<^{-1}$
$>$	?	$df<^{-1}$
$\leq$	$<$	<s
$\geq$	$<$	sd
$\geq$	$\leq$	sdf
$\geq$	?	$sdf<^{-1}$
?	$<$	<sd
?	$\leq$	<sdf
?	?	$<sdf<^{-1}$

3.2 Point Graphs (PG)

The inference mechanism of PIL is implemented with the help of a graph, called Point Graph (PG). The expressions in PIL are transformed to their PG representations, and the graph so constructed is processed before being used for the inferences. This section presents a detailed account of the PG representation and the graph operations applied to it.

Definition 3.2.1: Point Graph

A Point Graph, PG (V, E_A, D, T) is a directed graph with:

V: Set of vertices with each node or vertex $v \in V$ representing a point on the real number line. Two points pX and pY are represented as a composite point [pX;pY] if both are mapped to a single point on the line.

E_A: Union of two sets of edges: $E_A = E \cup E_{\leq}$, where

E: Set of edges with each edge $e_{12} \in$ E, between two vertices v1 and v2, also denoted as (v1, v2), representing a relation '<' between the two vertices—(v1 < v2). The edges in this set are called LT edges;

$E_{\leq}$: Set of edges with each edge $e_{12} \in E_{\leq}$, between two vertices v1 and v2, also denoted as (v1, v2), representing a relation '≤' between the two vertices—(v1 ≤ v2). The edges in this set are called LE edges.

D: Edge-length function (possibly partial): $E \rightarrow \Re^+$

T: Vertex-stamp function (possibly partial): $V \rightarrow \Re$

Figure 3.2.1 presents a three-node Point Graph with vertex stamps and arc length, and the corresponding PIL system represented by the PG. The figure also presents a correspondence between the stamps and edge lengths: a PG with only stamps can be represented by an equivalent PG with edge length expressions and vice versa by using a reference stamp for the conversion.

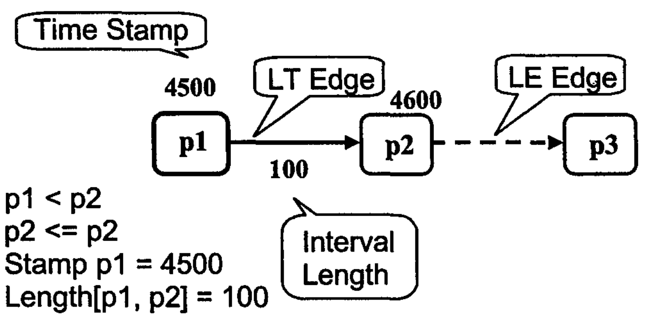


Figure 3.2.1. Point Graph Representation of a set of PIL Expressions

A relation Ri between two intervals X and Y can now be represented by an equivalent Point Graph representation by translating the algebraic inequalities shown in Tables 3.1.3-5 to corresponding PGs. Figure 3.2.2 illustrates the conversion with the help of some example PIL statements and their corresponding PGs.

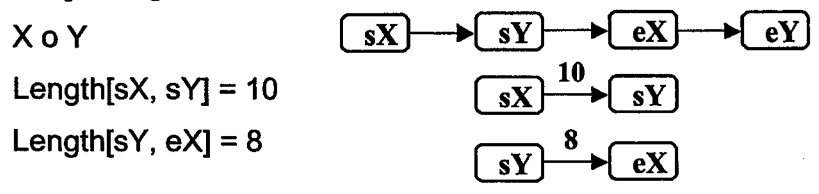


Figure 3.2.2. PIL Statement to PG Translation

The PG representing the entire system of PIL statement is then constructed by unifying (Definition 3.2.2, below) individual PGs to a (possibly) single connected graph. The unifying process only looks at the labels of the nodes to identify equalities, and does not take into consideration the arc lengths assigned to edges in the PG.

Definition 3.2.2: Pre-set (Post-set)

A pre-set (post-set) of a node contains all the nodes in V that have directed edges originating from (terminating at) them and terminating at (originating from) node v. The notation *v (v*) represents the pre-set (post-set) of a node v.

$\forall$ vi, vi $\in$ *v, then (vi, v) $\in$ E_A

Similarly,

$\forall$ vi, vi $\in$ v*, then (v, vi) $\in$ E_A

Definition 3.2.3: Unification

(a) Let vi = [pi;...;pn] and vj = [pj;...;pm] be two nodes in a PG representation. If there exists a point pk such that pk $\in$ [pi;...;pn] and pk $\in$ [pj;...;pm] or T(vi) = T(vj) then the two nodes are merged into a single composite node 'vi;vj' such that:

vi;vj = [pi;...;pn] $\cup$ [pj;...;pm]

*vi;vj = *vi $\cup$ *vj

vi;vj* = vi* $\cup$ vj*

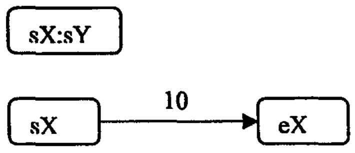
(The notation [pi;…,pn] represents a composite point.) The change in pre- and post-sets of unified nodes results in redefinition of the set E_A in the PG representation. The nature of the edges involved in the unification does not change in the redefinition.

T(vi;vj) = T(vi) = T(vj)

Example 3.2.1

a) sX = sY

Length[sX, eX] = 10



b) The point graph after unification:

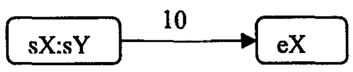


Figure 3.2.3. Unification I

or

(b) For all vi and vj ∈ V, s.t. T(vi) < T(vj,) construct a directed edge from node vi to vj with D(vi, vj) = T(vj) – T(vi);

(The corresponding sets V, E_A, and the functions D, T, are accordingly updated.)

Example 3.2.2

a) Time[sX] = 10

Time[sY] = 20

b) The point graph after unification:

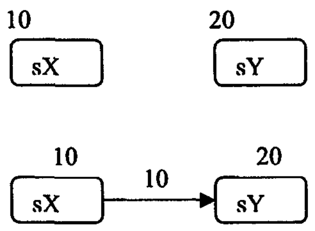


Figure 3.2.4. Unification II

The unified PG is then scanned for Join and Branch nodes (Definition 3.2.4, below) with quantitative information on their incoming and outgoing edges, respectively. The PG is then folded (Definitions 3.2.4-3.2.6, below) at these types of nodes. The folding process establishes new relations among system intervals, inferred through the quantitative analysis of the known relations specified by interval lengths and stamps.

Definition 3.2.4: Branch (Join) Node

A vertex v ∈ V in a Point Graph is termed a Branch (Join) node if it has multiple outgoing (incoming) edges connected to it.

Figure 3.2.5 shows a pictorial representation of a branch and a join node in Point Graphs.

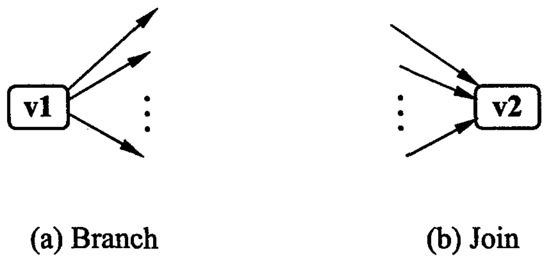


Figure 3.2.5. Branch and Join Nodes in Point Graphs

Definition 3.2.5: Branch Folding

A branch node vi ∈ V is said to be folded if, for all vj and vk in the post-set of vi, with:

(a) D(vi, vj) < D(vi, vk) the edge from vi to vk , denoted as (vi, vk), is replaced by an edge (vj, vk) with

D(vj, vk) = D(vi, vk) – D(vi, vj)

and the vertex vk removed from the post-set.

<u>Example 3.2.3</u>

a) Length[sX, eX] = 10

Length[sX, sY] = 20

$$\begin{tikzcd} sX \arrow[r, "10"] & eX \\ sX \arrow[r, "20"] & sY \end{tikzcd}$$

b) The point graph after unification:

$$\begin{tikzcd} & eX \\ sX \arrow[ur, "10"] \arrow[dr, "20"'] & \\ & sY \end{tikzcd}$$

c) The point graph after branch folding:

$$\begin{tikzcd} sX \arrow[r, "10"] & eX \arrow[r, "10"] & sY \end{tikzcd}$$

Figure 3.2.6. Branch Folding I

or

(b) D(vi, vj) = D(vi, vk), the two vertices vj and vk are merged into a single vertex with composite label 'vj;vk', and

D(vi, vj;vk) = D(vi, vk) {= D(vi, vj)}

<u>Example 3.2.4</u>

a) Length[sX, eX] = 10

Length[sX, sY] = 10

$$\begin{tikzcd} sX \arrow[r, "10"] & eX \end{tikzcd}$$

$$\begin{tikzcd} sX \arrow[r, "10"] & sY \end{tikzcd}$$

b) The point graph after unification:

$$\begin{tikzcd} & eX \\ sX \arrow[ur, "10"] \arrow[dr, "10"'] & \\ & sY \end{tikzcd}$$

c) The point graph after branch folding:

$$\begin{tikzcd} sX \arrow[r, "10"] & eX{:}sY \end{tikzcd}$$

Figure 3.2.7. Branch Folding II

or

(c) vi has multiple edges to vj. If the edges are all of the same type (LT or LE) then only one edge is retained and others are deleted. If at least one of them is of type LT then it is retained and others are deleted. If D(vi, vj) is defined for one of these edges, the value is assigned to the surviving edge.

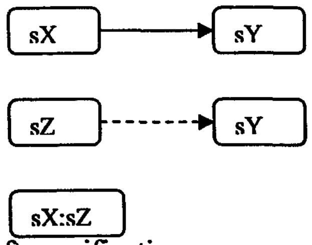
(The corresponding sets V, E_A, and the functions T, D are accordingly updated.)

<u>Example 3.2.5</u>

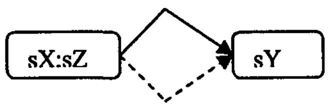
a) sX < sY

sZ <= sY

sX = sZ



b) The point graph after unification:



c) The point graph after folding:

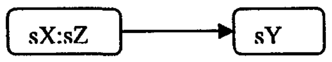


Figure 3.2.8. Branch Folding III

The methodology applies the branch-folding process to all the original and newly created (formed during the folding process) branch nodes in the unified net.

The branch folding process, when applied to all the branch nodes of a graph, yields a partially folded PG having nodes with at most one outgoing edge with edge length expression. Since all the edges in the PG may not have edge lengths associated with them, the branch folding may not result in a branch-node-free PG. A join folding process, which applies a similar process to all the joins in the graph, further treats the PG so obtained.

<u>Definition 3.2.6</u>: Join Folding

A join node vi $\in$ V is said to be folded if, for all vj and vk in the pre-set of vi, with:

(a) D(vj, vi) < D(vk, vi), the edge (vk, vi), is replaced by an edge (vk, vj) with

D(vk, vj) = D(vk, vi) – D(vj, vi)

and the vertex vk removed from the pre-set.

Example 3.2.6

a) Length[sX, eX] = 10

Length[sY, eX] = 20

$$\begin{tikzcd}
\boxed{sX} \arrow[r, "10"] & \boxed{eX} \\
\boxed{sY} \arrow[r, "20"] & \boxed{eX}
\end{tikzcd}$$

b) The point graph after unification:

$$\begin{tikzcd}
\boxed{sX} \arrow[dr, "10"] & \\
& \boxed{eX} \\
\boxed{sY} \arrow[ur, "20"'] &
\end{tikzcd}$$

c) The point graph after join folding:

$$\begin{tikzcd}
\boxed{sY} \arrow[r, "10"] & \boxed{sX} \arrow[r, "10"] & \boxed{eX}
\end{tikzcd}$$

Figure 3.2.9. Join Folding I

or

(b) D(vj, vi) = D(vk, vi), the two vertices vj and vk are merged into a single vertex with composite label 'vj;vk', and

D(vj;vk, vi) = D(vk, vi) {= D(vj, vi)}

Example 3.2.7

a) Length[sX, eX] = 10

Length[sY, eX] = 10

$$\begin{tikzcd}\boxed{sX} \arrow[r, "10"] & \boxed{eX}\end{tikzcd}$$

$$\begin{tikzcd}\boxed{sY} \arrow[r, "10"] & \boxed{eX}\end{tikzcd}$$

b) The point graph after unification:

$$\begin{tikzcd}\boxed{sX} \arrow[dr, "10"] & \\ & \boxed{eX} \\ \boxed{sY} \arrow[ur, "10"'] & \end{tikzcd}$$

c) The point graph after join folding:

$$\begin{tikzcd}\boxed{sX{:}sY} \arrow[r, "10"] & \boxed{eX}\end{tikzcd}$$

Figure 3.2.10. Join Folding II

or

(c) vi has multiple edges from vj. If the edges are all of the same type (LT or LE) then only one edge is retained and others are deleted. If at least one of them is of type LT then it is retained and others are deleted. If D(vj, vi) is defined for one of these edges, the value is assigned to the surviving edge.

(Note: Here the case c is redundant because this case has already been taken care of during branch folding.)

(The corresponding sets V, E_A, and the functions T, D are accordingly updated.)

A single application of join folding after a single application of branch folding is all that is needed to fully fold the graph. A proposition by Zaidi and Levis, in (2001), ensures the fact that single applications of branch folding followed by join folding are enough to fold the graph completely (the term 'completely' is used relative to the quantitative information available in the PG.)

Figure 3.2.11 illustrates the process of converting a set of PIL statements to their PG representation (Figure 3.2.11a, and 3.2.11b.) The figure also shows the result of the unification of the PG (Figure 3.2.11c.) The join- and branch-folding of the PG are shown in parts (d) and (e), Figure 3.2.11.,

(a) Set of PIL Statements

X, Y, Z intervals

X o Y

Length [sX, sY] = 10

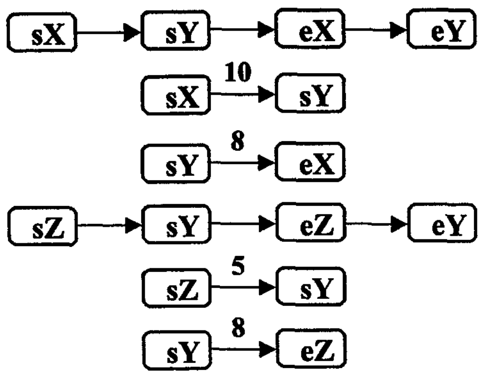
Length [sY, eX] = 8

Z o Y

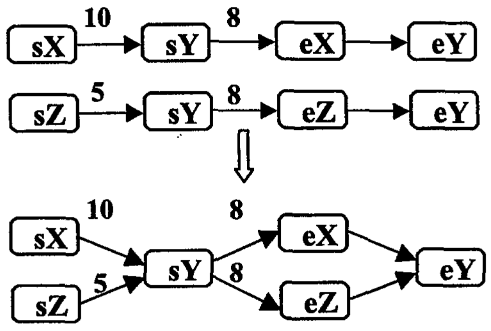
Length [sZ, sY] = 5

Length [sY, eZ] = 8

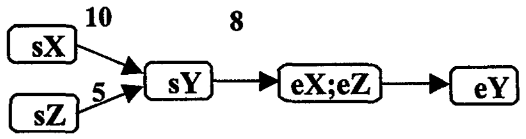
(b) Construction of Point Graph



(c) Unification and Resulting PG



(d) Branch Folding



(e) Join Folding

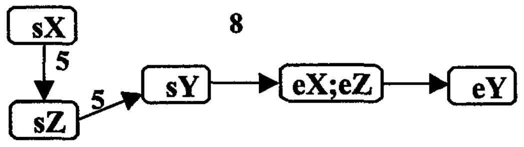


Figure 3.2.11. Steps in PG Construction

The PG representation of PIL statements helps the inference mechanism of PIL to construct the string representation for the pairs of intervals (Tables 3.1.3-5) with unknown relations by performing a simple search in the PG constructed after unification and folding processes. The existence of a directed path from a node 'p' to another 'q' with at least one LT edge in it establishes the relation 'p < q' between the two points. A path between the two nodes with only LE type edges establishes the relation '$p \leq q$' between the two. An inference for a PIL relation between two intervals requires at most eight searches to be performed, two for each pair of start/end points. The resulting string representation is pattern matched with the strings in Tables 3.1.3-5 to identify the corresponding atomic/compound PIL relation. As mentioned earlier, an inference resulting in a compound relation of the type $RiRjRk^{-1}$ between two intervals X and Y represents the disjunction of 'X Ri Y', 'X Rj Y', and 'Y Rk X'. The search for the directed path between two vertices in a PG uses a depth-first search with arc lengths as the heuristic measure; the depth-first search engine first explores the outgoing edge of the current vertex with a length expression. The search, therefore, finds the path between two vertices that has (possibly) all its constituent edges with length expressions. The sum of all these lengths gives the total distance between the two vertices (points). Similarly, if the stamp of one of these points is known, the stamp of the other can be calculated by adding or subtracting the distance (path length) between the two.

The illustration in Figure 3.2.11 shows the new PIL relations that can be inferred for the PIL system modeled by the approach. The PIL statements 'Z f X' and 'Length [sX, sZ] = 5' can easily be inferred through the PG in Figure 3.2.11e.

3.3. Verification of PIL Statements

The inference mechanism described in the previous section may result in erroneous and inconsistence results provided the system of PIL statements, represented by the PG, contains *inconsistent* information. The inference, on the other hand, is guaranteed to yield valid assertions given a consistent PIL system and corresponding PG representation. This section characterizes the inconsistencies in a PIL system and in its PG representation. The section also presents methods to verify a PIL system for these erroneous instances.

Definition 3.3.1 Inconsistency

A set of statements (inferences) is said to be inconsistent if the statements in the set cannot all be true at the same time.

Definition 3.3.2: Interpretation Function, I_f

An interpretation function in PIL assigns a Stamp to each node in the PG representation of a system of PIL statements.

I_f: $V_\Delta \rightarrow \Re$, where Δ is a system of PIL statements

Definition 3.3.3: Interpretation, I

An interpretation of a system of PIL statements Δ assigns a 'True' or 'False' value to each atomic PIL statement in Δ.

I: $\Delta \rightarrow$ {True, False}

Theorem 3.3.1

For any interpretation function for a set of PIL statements Δ, there is a unique interpretation of Δ.

The theorem follows from the definitions of interpretation function, interpretation, and the analytical representation of PIL relations given in Tables 3.1.3-5.

Definition 3.3.4: Satisfaction, Model

A system of PIL statements Δ is satisfied by an interpretation I iff I(Δ) = True, also denoted as $\models_I \Delta$. An interpretation satisfying a system of PIL statements Δ is called a Model of Δ.

Theorem 3.3.2

A system of PIL statements Δ is inconsistent if and only if it is unsatisfiable, i.e., there does not exist an interpretation that satisfies Δ.

The theorem directly follows from the definition of inconsistency (Definition 3.3.1) and definition of an interpretation of a system of PIL statements. The definition of inconsistency in Definition 3.3.1 and Theorem 3.3.2, and Proposition 3.3.1 lead to another (operational) characterization of inconsistency in Theorem 3.3.3. The theorem is an extension of an earlier result presented by Zaidi and Levis (2001).

Theorem 3.3.3: Inconsistency in PIL

A system's description in PIL contains inconsistent information iff

(a) for some intervals X and Y, and atomic PIL relations Ri and Rj, both 'X Ri Y' and 'X Rj Y', $i \neq j$, or 'X Ri Y' and 'Y Ri X' (with the exception of = relation) hold true;

or

(b) for some intervals and/or points, the system can determine two string representations such that at least one pair of the algebraic inequalities representing relationships between the corresponding points represents an inconsistency. Let the two string representations be 'abcd' and 'uvwx', where a, b, c, d, u, v, w, and x ∈ {<, =, >, ≤, ≥,?}. One of the (un-ordered) pairs of corresponding inequalities, i.e.,

(a, u), (b, v), (c, w), or (d, x) ∈ {(<, =), (<, >), (<, ≥), (=, >), (>, ≤)};

or

(c) for a point p1, the system calculates two different stamps;

or

(d) for some points p1 and p2, 'p1 < p2', the system can determine two different lengths for the interval [p1, p2].

The part (a) of the theorem entails part (b), but not vice versa. It is therefore imperative to look for the cases described by part (b) for identification of inconsistent PIL statements. Some of the inconsistent cases, of the type defined in the other two parts (c) and (d), are trivially detected during the unification process: whenever two nodes with different stamps are merged into a single node—an inconsistency.

Once a unified Point Graph representation is achieved, the graph is checked for other inconsistent cases defined by the part (b) in the theorem. Such inconsistent cases are characterized by the following theorem.

<u>Theorem 3.3.4</u>

A set of PIL statements is inconsistent if the PG representation of the set contains self-loops and/or cycles with some LT type edges involved in the cycles.

A necessary condition for a consistent set of temporal statements is, therefore, given as:

<u>Theorem 3.3.5</u>

A set of temporal statements is consistent only if the PG representation of the set is an acyclical graph.

<u>Proofs</u>:

A system of PIL statements is inconsistent if two (or more) *different* PIL relations can be established and/or inferred between two intervals X and Y. The corresponding string representation (Tables 3.1.3-5) for the two relations would mean that two different inequalities could be established between at least on pair of ending points (e.g., p1 and p2) of the two intervals. In PG representation, the two different inequalities would result in two paths, one from p1 to p2 and the other from p2 to p1. The two paths together form a cycle.

The verification mechanism of PG representation identifies these inconsistent cases by applying the following result (Theorem 3.3.6)

Theorem 3.3.6

A point graph contains cycles if and only if it has non-zero S-invariants calculated for the Connectivity Matrix of the Point Graph.

Definition 3.3.5: Connectivity Matrix

A Point Graph with n directed edges and m nodes can be represented by a (n × m) matrix J, the Connectivity matrix. The rows correspond to edges and the columns correspond to nodes.

- $j_{ij} = 1$ if the directed edge in *i*th row originates from the *j*th node,
- $j_{ij} = -1$ if the directed edge in *i*th row terminates in the *j*th node,
- $j_{ij} = 0$ if the directed edge in *i*th row is not connected to *j*th node.

Note that in constructing the Connectivity Matrix no distinction is made between LT and LE type edges.

Definition 3.3.6: S-Invariant

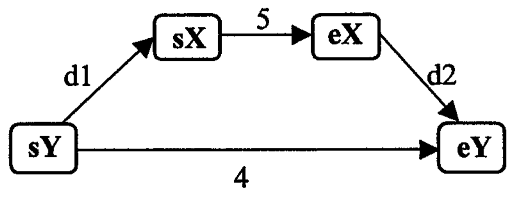
Given the Connectivity Matrix J of a Point Graph, an S-invariant is an n × 1 non-negative integer vector X of the kernel of J^T, i.e.,

$$J^T X = 0$$

The verification approach, therefore, constructs a Connectivity Matrix of the unified PG and calculates the S-invariants of the graph. The S-invariants can be calculated using an improved version of Farkas algorithm by Martinez and Silva, 1982; Memmi and Vautherin, 1987]. The resulting non-zero S-invariants identify the cycles (inconsistencies) in the system. Once cycles are detected in a PG by calculating non-zero S-invariants, the nodes responsible for these cycles can be easily identified. This will, in turn, identify intervals involved in these cycles. This information can be used to correct the system of PIL statements.

The folding process (Definitions 3.2.5 and 3.2.6) establishes new PIL relations, among system intervals, inferred through the quantitative analysis of the known relations specified by interval lengths and stamps. The possible inconsistencies present in the quantitative inputs may hinder the folding process or result in erroneous structures [Zaidi and Levis, 2001] of the folded graph. The type of inconsistency defined by Theorem 3.3.3d may reveal itself during the folding process: if during folding a PG the process finds multiple edges between a branch(join) node and a vertex in its post-(pre-)set, where these edges have different lengths associated with them, then the process halts and reports an error. The inconsistency can also result in creation of new cycles in the graph during the folding process. These cycles can be identified using another application of S-invariant algorithm.

The creation of cycles during the folding process can have serious effects on the graph. Once a cycle is created during the folding process, it tends to attract the remaining vertices in the PG towards itself. And if the PG has edge lengths on all its edges, the folding process ends up with a folded PG, which has a single cycle with all its vertices collapsed into it. The phenomenon is termed 'Black Hole Effect.' The intensive computational effort required in folding a PG, and a subsequent loss of it due to the black hole effect demand an earlier detection of cycles during the folding process itself. The folding procedure is, therefore, tailored to identify cycles by assigning dummy time stamps to vertices being folded: reassignment of a time stamp to an already marked vertex prompts the presence of a cycle. A folded PG with leftover branch and join nodes should also be checked for multiple directed paths from any branch node to any other join node. The length expressions corresponding to each such path are equated to each other and the resulting set of equations is checked for feasibility. A set of infeasible equations signals an inconsistent case present in the system. Figure 3.3.1 presents an example of such an inconsistent case.



$$d1 + 5 + d2 = 4, \quad d1, d2 > 0$$

Figure 3.3.1. An Example Inconsistent Case

An alternate technique to S-invariant algorithm, called path-searching algorithm, by Ma (1999) uses the *adjacency* and *reachability* matrices of the PG representation to uncover the cycles and inconsistent multiple paths between pairs of nodes (Theorem 3.3.3d.) The path-searching algorithm employs techniques by Busacker and Saaty (1965) and Warshall's algorithm to identify the erroneous cases.

Path Searching Algorithm

Path-searching algorithm is based on the *adjacency matrix* and the *reachability matrix* in graph theory. It has several advantages over the version using s-invariant concept to check if a Point Graph contains cycles or not [Zaidi and Levis, 1998]. One way to calculate the S-invariant is to use the Farkas algorithm modified by several researchers [Martinez and Silva, 1982; Memmi and Vautherin, 1987]. Usually, the *connectivity matrix* has to be utilized. The dimension of this matrix is $n \times m$, with n referring to the number of connectors and m referring to the number of nodes in the point graph, respectively. It is apparent that n will be greater than m in most situations. In the Farkas algorithm, an identity matrix is used to help get the result. The total storage requirement of the algorithm is therefore $n \times (n+m)$. This is larger than that for the adjacency matrix and the reachability matrix, which is only $m \times m$.

The second advantage of the new algorithm is that it determines if cycles exist before searching for the exact cycles. This makes the verification process a quick one if no cycles exist. Thirdly, the new algorithm can be applied to both the detection of cycles and the detection of inconsistent paths. This results in a simpler methodology. Finally, the new algorithm is more straightforward than the one based on Petri Net theory.

Definition 3.3.7: Adjacency (or Vertex) Matrix:

Given a graph or digraph with vertices indexed as $V(G) = \{v_1, \cdots, v_n\}$, the adjacency matrix of G, written as *A(G)*, is the matrix in which entry a_{ij} is the number of edges incident with both vertex *i* and vertex *j* (or directed from vertex *i* to vertex *j* in the digraph case).

[A vertex *v* is *incident* with an edge *e* if $v \in e$.]

Example 3.3.1

A Point Graph is in fact a digraph. The node of the point graph is the vertex of the digraph, and the connectors are edges. Figure 3.3.2 shows a very simple Point Graph to give an example on the adjacency matrix calculation. The abstract notation e_i (i=1..5) is used for explanation convenience, it can be replaced by the actual time durations for the edges.

Adjacency matrix for Figure 3.3.2 is as follows:

$$A = \begin{array}{c|ccccc} & v_1 & v_2 & v_3 & v_4 & v_5 \\ \hline v_1 & 0 & 1 & 1 & 0 & 0 \\ v_2 & 0 & 0 & 0 & 0 & 1 \\ v_3 & 0 & 0 & 0 & 1 & 0 \\ v_4 & 0 & 0 & 0 & 0 & 1 \\ v_5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The powers of matrix A are given as:

$$A^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad A^4 = A^5 = \vec{0}_{5\times 5}$$

The elements a_{15}^1, a_{15}^2, a_{15}^3, a_{15}^4, a_{15}^5 in each corresponding A^i matrix have the following meanings:

- $a_{15}^1 = 0$ means there is no path from v_1 to v_5 with the path length equals to 1.
- $a_{15}^2 = 1$ means there is only one path from v_1 to v_5 with the path length equals to 2, which is (v_1, v_2, v_5).
- $a_{15}^3 = 1$ means there is only one path from v_1 to v_5 with the path length equals to 3, which is (v_1, v_3, v_4, v_5).
- $a_{15}^4 = 0$, $a_{15}^5 = 0$ mean that there are no paths from v_1 to v_5 with path lengths equals to 4 and 5 respectively.

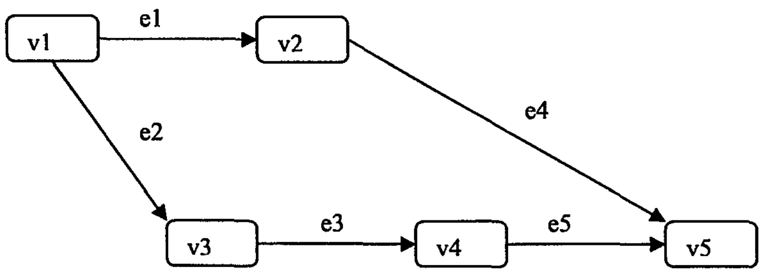


Figure 3.3.2. A digraph with PG structure

Proposition 3.3.1: (Busacker and Saaty, 1965)

The matrix A^n gives the number of arc progressions of length n between any two vertices of a directed graph.

Proof:

If a_{ik} is the number of arcs joining v_i to v_k and a_{kj} is the number of arcs joining v_k to v_j, then $a_{ik}a_{kj}$ is the number of different paths each consisting of two arcs joining v_i to v_j and passing through v_k. If this is summed over all values of k, that is, over all the intermediate vertices, one obtains the number of paths of length 2 between v_i and v_j. If we now use a_{ij} to form $a_{ij}a_{jm}$, we have the number of different paths of length 3 between v_i and v_m passing through v_j, and so on. Thus if we assume the theorem true for A^{n-1}, then the coefficients of $A^n = A^{n-1}A$ give the number of paths of length n between corresponding vertices. This completes the proof.

Observe another matrix calculated by adding the powers of A:

$$B = A + A^2 + A^3 + \cdots + A^n$$

For the above example,

$$B = A + A^2 + A^3 + A^4 + A^5 = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In this B matrix, $b_{15} = 2$ means there are two paths between v_1 and v_5. If we only care whether v_i is reachable from v_j, we can make a modification to matrix B to describe the reachability between any nodes in a digraph G (also Point Graph).

Definition 3.3.8: Reachability Matrix:

Given a simple digraph G = (V, E) with vertices indexed as $V(G) = \{v_1, \cdots, v_n\}$, the matrix R with rank n, $R = (r_{ij})_{n\times n}$, is called the reachability matrix, where

$$r_{ij} = \begin{cases} 1 & b_{ii} \neq 0 \quad \text{(there is at least one walk between } v_i \text{ and } v_j\text{)} \\ 0 & \text{otherwise} \quad \text{(there is no walk between } v_i \text{ and } v_j\text{)} \end{cases}$$

Especially,

$$r_{ii} = \begin{cases} 1 & \text{there is a cycle from } v_i \text{ to } v_i \\ 0 & \text{otherwise} \end{cases}$$

One simple method to calculate matrix R is to derive it from matrix B directly. The reachability matrix for Figure 3.3.2 is shown below. However, this method is computationally intensive. It has to calculate A^2, A^3, …, A^n, and then add them together with *A*. Finally, change the non-zero elements a_{ij} to the value 1.

$$R = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Actually, matrix R is a Boolean Matrix. So we can adapt matrix addition and multiplication to Boolean addition and Boolean multiplication. Warshall's algorithm realizes the above idea.

<u>Algorithm 3.3.1:</u> Warshall Algorithm [Warshall, 1962]

Given a graph G = (*V*, *E*) with the vertices indexed as V = {1, 2, 3, ···, n}, construct a matrix series: A = $R^{(0)}$, $R^{(1)}$, $R^{(2)}$, ···, $R^{(n)}$ = R, here $R^{(0)}$ is the adjacency matrix, and $R^{(n)}$ is the reachability matrix. The basic idea for the Warshall algorithm is explained as follows:

Step 1 Calculate $R^{(0)}$ from A

$r_{ij}^{(0)} := a_{ij}$

Here, $r_{ij}^{(0)}$ is the element of $R^{(0)}$, a_{ij} is the element of A.

Step 2 Calculate $R^{(1)}$ from $R^{(0)}$

$r_{ij}^{(1)} := r_{ij}^{(0)} \vee (r_{i1}^{(0)} \wedge r_{1j}^{(0)})$

The above expression has two meanings:

1) if an edge exists directly from vertex i to vertex j, then $r_{ij}^{(1)} = 1$;
2) if an edge exists directly from vertex i to vertex 1, and also an edge exists directly from vertex 1 to vertex j, which means vertex i, j is connected via vertex 1, then $r_{ij}^{(1)} = 1$.

Step 3 Calculate $R^{(2)}$ from $R^{(1)}$

$r_{ij}^{(2)} := r_{ij}^{(1)} \vee (r_{i2}^{(1)} \wedge r_{2j}^{(1)})$

The above expression has two meanings:

1) if $r_{ij}^{(1)} = 1$, which means vertex i, j have a direct edge between them or they are connected via vertex 1, then $r_{ij}^{(2)} = 1$;
2) if $r_{i2}^{(1)} = 1$ (vertex i and vertex 2 have a direct edge between them or they are connected via vertex 1) and $r_{2j}^{(1)} = 1$ (vertex 2 and vertex j have a direct edge between them or they are connected via vertex 1), then $r_{ij}^{(2)} = 1$.

From the above, it is apparent:

$r_{ij}^{(2)} = 1 \Leftrightarrow$ vertex i, j has a direct edge connected or connected via vertex 1, 2.

...

Step k Calculate $R^{(k)}$ from $R^{(k-1)}$

$r_{ij}^{(k)} := r_{ij}^{(k-1)} \vee (r_{i2}^{(k-1)} \wedge r_{2j}^{(k-1)})$

The above expression means:

$r_{ij}^{(k)} = 1 \Leftrightarrow$ vertex i, j can be connected via vertex 1, 2, ⋯, k.

...

Step n Calculate $R^{(n)}$ from $R^{(n-1)}$

$r_{ij}^{(n)} := r_{ij}^{(n-1)} \vee (r_{i2}^{(n-1)} \wedge r_{2j}^{(n-1)})$

The above expression means:

$r_{ij}^{(k)} = 1 \Leftrightarrow$ vertex i, j can be connected via no more than n vertexes.

According to the above analysis, Warshall's Algorithm can be described as:

```
Step 1:  $R := A$
Step 2:  $k := 1$
Step 3:  $i := 1$
Step 4:  for $j:=1$ to $n$ do $r_{ij} := r_{ij} \vee (r_{ik} \wedge r_{kj})$
Step 5:  $i := i+1$; if $i \leq n$ then go to step 4
Step 6:  $k := k+1$; if $k \leq n$ then go to step 2
Step 7:  end
```

The algorithm can be expressed in the form of a flow chart, as shown in Figure 3.3.3.

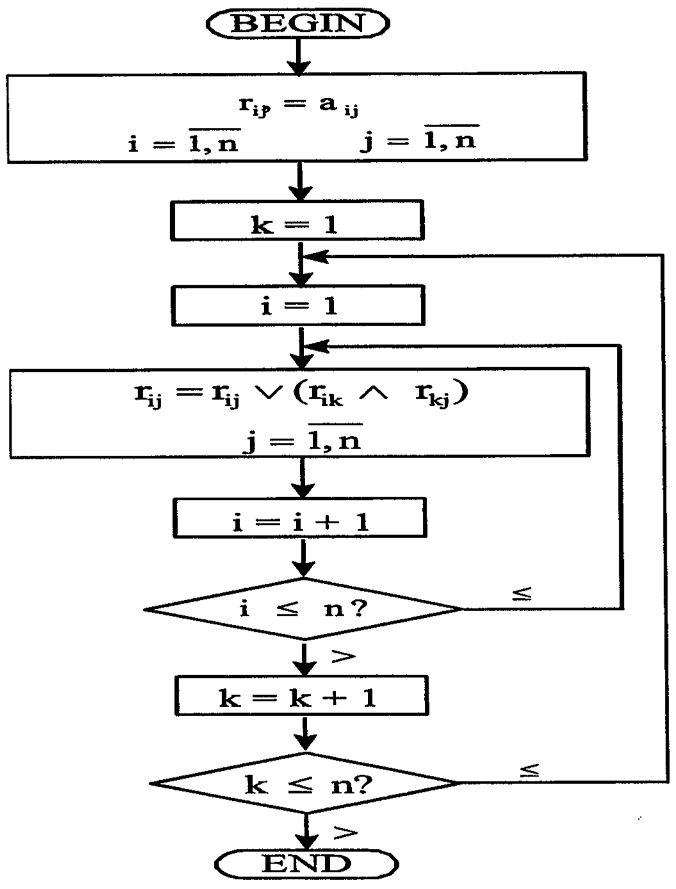


Figure 3.3.3. Flow Chart for Warshall's Algorithm

Warhshall's algorithm provides an efficient way to calculate the Reachability Matrix based on the Adjacency Matrix. Using the reachability matrix, checking the connectivity between any two nodes v_i and v_j becomes very easy. $r_{ij} = 1$ means there is at least one path from v_i to v_j; otherwise, $r_{ij} = 0$ means there is no path from vi to v_j. If a node v_i is involved in a cycle, the element r_{ii} in R must take the value of 1. Thus, if there are non-zero elements on the diagonal line of matrix R for a Point Graph, this Point Graph has cycle structures in it. Otherwise, we can conclude there is no cycle in this Point Graph. This result avoids the search for cycles when none exist.

<u>Algorithm 3.3.2:</u> Path-Searching Algorithm

The path-searching algorithm can be applied on both cycle detection and inconsistent path detection. A path is a connected chain having a starting node and an ending node with zero, one or more interim nodes between them. A cycle is a special case of the path with the starting node and the ending node being the same one. An example is used to illustrate how the path-searching algorithm works for finding out multiple paths between a branch node and a join node.

Example 3.3.2

Consider the branch node v_1 and the join node v_5 in Figure 3.3.2 and find out all the paths from v_1 to v_5:

Step 1: Check out the connectivity from node v_1 to v_5 using matrix R. Since $r_{15} = 1$, there is at least one path from v_1 to v_5.

Step 2: Obtain the row v_1 from matrix A. Assign the value to $X^{(1)}$, i.e., $X^{(1)} = [0, 1, 1, 0, 0]$. Since $x^{(1)}_{15} = 0$, it means the path from v_1 to v_5 with length 1 is 0.

Step 3: Calculate the elements of $X^{(2)}$ via the dot product of $X^{(1)}$ with each column of matrix A. $X^{(2)} = [0, 0, 0, 1_{(v3)}, 1_{(v2)}]$. Since $x^{(2)}_{15} = 1$, the path from v1 to v5 with length 2 is 1. The node that makes the contribution to this connection is v_2, which can be obtained by Boolean 'AND' operation between $X^{(1)}$ and the v_5 column in matrix A. The path with length 2 from v_1 to v_5 is composed of nodes 'v_1, v_2, v_5'.

Step 4: Calculate the elements of $X^{(3)}$ via the dot product of $X^{(2)}$ with each column of matrix A. $X^{(3)} = [0, 0, 0, 0, 1_{(v4)}]$. Since $x^{(3)}_{15} = 1$, the path from v1 to v5 with length 3 is 1. The nodes that make the contribution to this connection are checked in this way: firstly, the Boolean 'AND' operation between $X^{(2)}$ and the v_5 column [0, 1, 0, 1, 0] yields [0, 0, 0, 1, 0], which means node v4 is most adjacent to v_5; secondly, yields a Boolean 'AND' operation between $X^{(1)}$ and the v_4 column of matrix A will get [0, 0, 1, 0, 0], which means node v_3 is most adjacent to v_4. Thus, the path with length 3 from v_1 to v_5 is composed of nodes 'v_1, v_3, v_4, v_5'.

Step 5: Calculate the elements of $X^{(4)}$ via the dot product of $X^{(3)}$ with each column of A. $X^{(4)} = [0, 0, 0, 0, 0]$. No more steps are necessary.

The above steps can be generalized to form the path-searching algorithm, which is shown as two flow charts (Figures 3.3.4 and 3.3.5).

The first part of the algorithm finishes the preparatory work. For two connecting nodes v_i and v_j, i and j are their indexes in matrix A, respectively. The algorithm collects all $X^{(k)}$ values for node v_i until $X^{(k)}$ becomes a zero vector or k increases to n (where n is the number of node in a Point Graph). The first part returns a set of $X^{(k)}$ values and the value of k.

In the second part, the algorithm utilizes the value of k and the $X^{(k)}$s to collect all the paths between node v_i and v_j. The element $x_{ij}^{(m)}$ is the jth element of $X^{(m)}$, where $1 \leq m \leq k$. The value of $x_{ij}^{(m)}$ denotes the number of paths characterized by m connectors in each path. The variable 'count' records how many times we need to split a path. The algorithm finally returns the total numbers of paths between the two nodes and the set of interim nodes involved in each path.

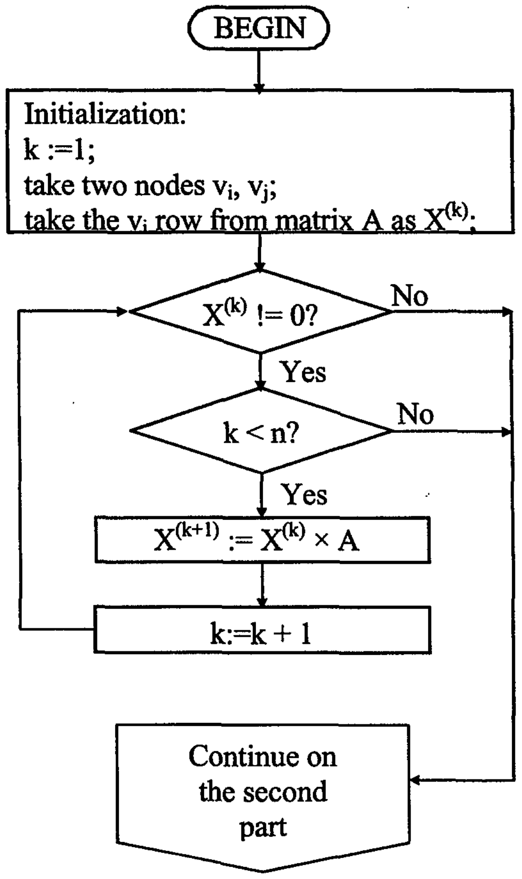

BEGIN
Initialization:
k :=1;
take two nodes v_i, v_j;
take the v_i row from matrix A as $X^{(k)}$;
$X^{(k)}$!= 0?
No
Yes
k < n?
No
Yes
$X^{(k+1)} := X^{(k)} \times A$
k:=k + 1
Continue on the second part

Figure 3.3.4. Path-Searching Algorithm (Part I)

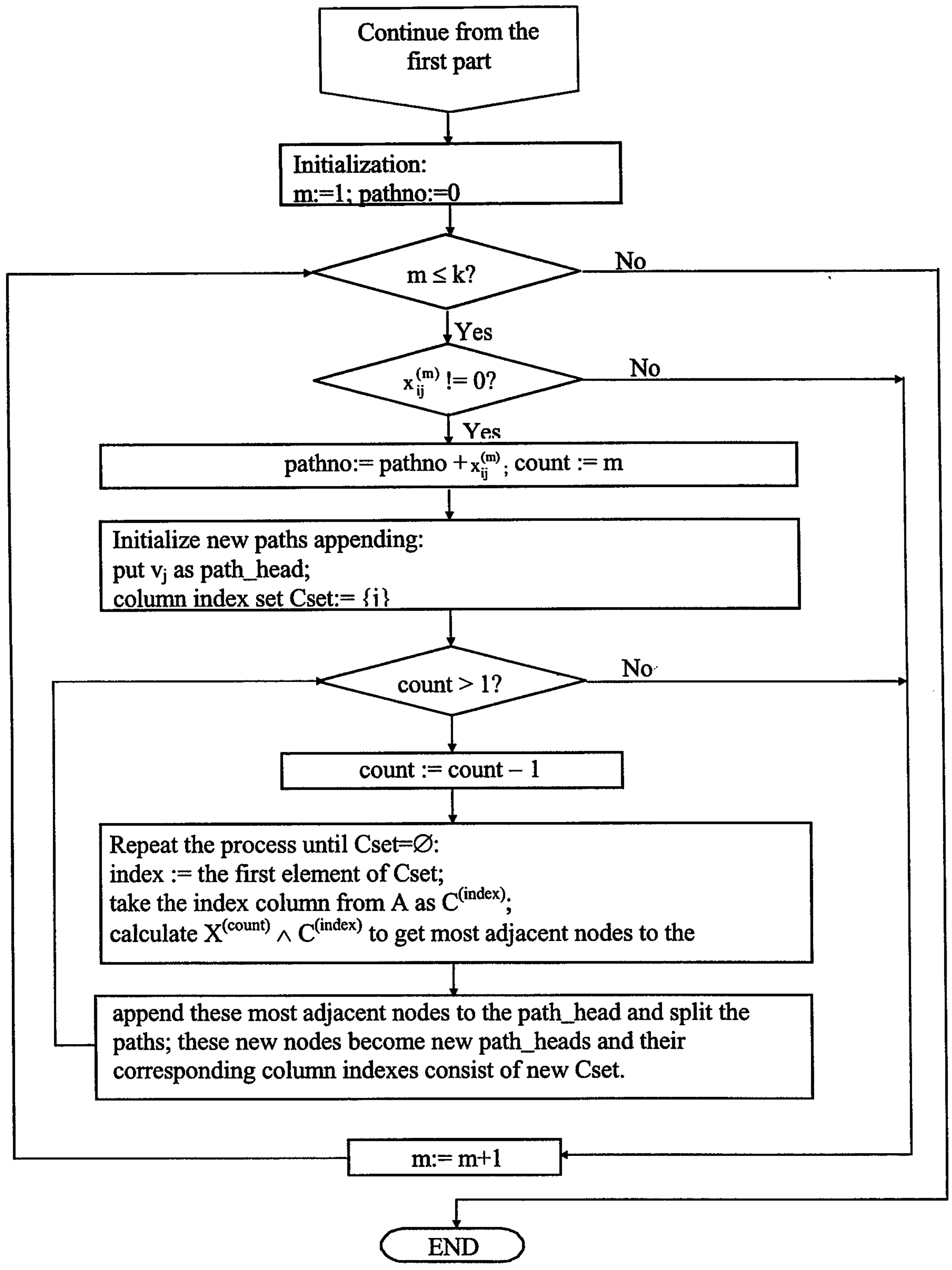

Continue from the first part
Initialization:
m:=1; pathno:=0
m ≤ k?
No
Yes
$x_{ij}^{(m)}$!= 0?
No
Yes
pathno:= pathno + $x_{ij}^{(m)}$; count := m
Initialize new paths appending:
put v_j as path_head;
column index set Cset:= {i}
count > 1?
No
count := count – 1
Repeat the process until Cset=∅:
index := the first element of Cset;
take the index column from A as $C^{(index)}$;
calculate $X^{(count)} \wedge C^{(index)}$ to get most adjacent nodes to the
append these most adjacent nodes to the path_head and split the paths; these new nodes become new path_heads and their corresponding column indexes consist of new Cset.
m:= m+1
END

Figure 3.3.5. Path-Searching Algorithm (Part II)

In the algorithm, if $v_i = v_j$, the paths detected between v_i and v_j will be cycles. That is how the algorithm can be applied on detecting cycle structures in a Point Graph. When we are checking inconsistent paths in a Point Graph, we will check the multiple directed paths from any branch node to any join node. Here, v_i will serve as a branch node, and v_j a join node.

Algorithm 3.3.3: Detection of Cycles

The methodology using the path-searching algorithm for detecting all cycles in a Point Graph is given as follows:

1) Construct C_n, the set of all cycle nodes by checking the non-zero diagonal elements on the matrix R.
2) If there are only two nodes in C_n, these two nodes contribute a cycle structure. Report the cycle and exit. Otherwise, go to step 3.
3) Select a node $c_i \in C_n$ and apply the path-search algorithm on node c_i and c_i. Remove from C_n all the nodes calculated from this path-searching process. If no elements are left in C_n, report the cycle(s) and exit, else go to step 2.

Figure 3.3.6 shows a Point Graph with a cycle. The reachability matrix R for this point graph is shown below. Since the diagonal elements on the matrix R are not all zeros, the existence of the cycle is detected.

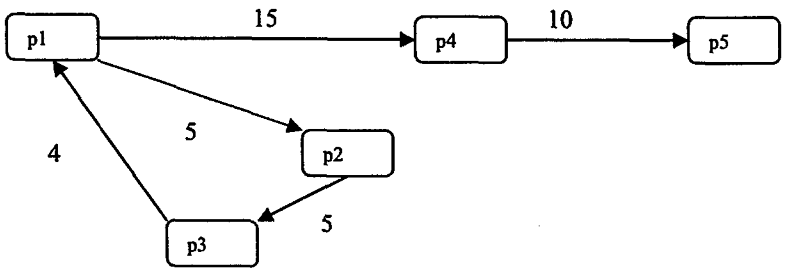


Figure 3.3.6. A Point Graph with Cycle Structure

Reachability Matrix for Figure 3.3.6 is as follows:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Algorithm 3.3.4: Detection of Inconsistent Paths

To find out all the inconsistent paths in a Point Graph, we summarize the following steps to apply the path-searching algorithm:

1) Construct V_B, the set of all branch nodes in the PG. Select a node $v_i \in V_B$ and remove v_i from V_B.
2) Construct V_J, the set of all join nodes in PG. Select a node $v_j \in V_J$ and remove v_j from V_J.
3) Determine the reachability between v_i and v_j by checking the element r_{ij} in the R matrix. If $r_{ij} = 0$, iterate through step 2 until there are no elements left in the set V_J. If $r_{ij} = 1$, apply the path-searching algorithm to find all directed paths from v_i to v_j. For any two paths p1 and p2 with lengths d1 and d2 respectively use the table 4.1 to detect inconsistent paths. Here if the path p contains LT edges then the flag LT is true for p, same is the case with LE edges. Iterate through step 2 until there are no elements left in the set V_j.
4) Go to step 1 until there are no elements left in the set V_B.

Table 3.3.1. Detection of Inconsistent Paths

LE	LT	LE	LT	ACTION
T	T	T	T	Consistent
T	T	T	F	Consistent but if p2 contains only single LE edge and d2 <= d1 change it into a LT edge
T	T	F	T	Consistent
T	T	F	F	If (d2 > d1) then Consistent else Inconsistent
T	F	T	T	Consistent but if p1 contains only single LE edge and d1 <= d2 change it into a LT edge
T	F	T	F	Consistent. If d1 < d2 and p1 contains only single LE edge change it into a LT edge else if d2 < d1 and p2 contains only single LE edge change it into a LT edge
T	F	F	T	Consistent but if p1 contains only single LE edge and d1 <= d2 change it into a LT edge
T	F	F	F	If (d1 > d2) then Consistent and also see if there is single LE edge in that case change it into a LT edge else if (d1 == d2) then Consistent but merge all LE edges else Inconsistent
F	T	T	T	Consistent
F	T	T	F	Consistent but if p2 contains only single LE edge and d2 <= d1 change it into a LT edge
F	T	F	T	Consistent
F	T	F	F	If (d2 > d1) then Consistent else Inconsistent
F	F	T	T	If (d1 > d2) then Consistent else Inconsistent
F	F	T	F	If (d1 > d2) then Consistent and also see if there is single LE edge in that case change it into a LT edge and set length = d1 – d2 else if (d1 == d2) then Consistent but merge all LE edges else Inconsistent
F	F	F	T	If (d1 > d2) then Consistent else Inconsistent
F	F	F	F	If (d1 == d2) then Consistent else Inconsistent

Figure 3.3.7 shows a Point Graph used to explain the "Black-Hole-effect" [Zaidi and Levis, 1998]. If the above methodology is applied to this Point Graph, before undertaking the folding process, it will detect the inconsistent paths.

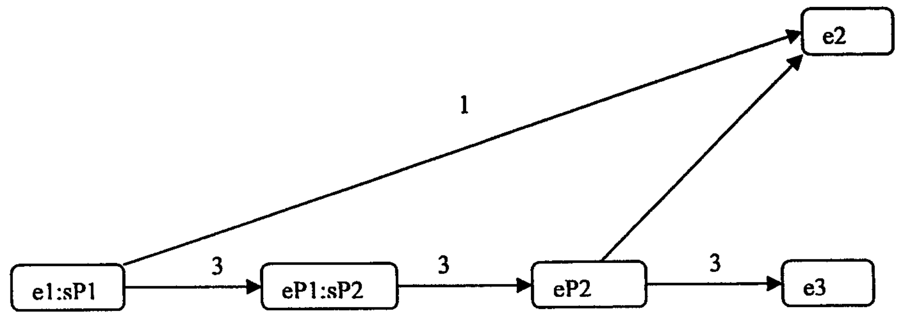


Figure 3.3.7. Inconsistent Paths in a Point Graph

3.4. Revision of PIL Systems

In real world domains, many situations require revising a produced model during and/or after system specification phase, e.g., the constraints or system/mission requirements may change during or before a plan's execution. The approach presented so far does not support revision (add, modify, delete); even a minor change in the input PIL system of statements can not be made unless one re-edited the statements and restarted the whole process from scratch. The changes in the input specifications are classified into the following three types (Figure 3.4.1) based on the impact of the change on the original set. [Ma, 1999]

Type 1—Local Change: Change that requires minor modifications of very few directly affected input statements;

Type 2—Regional Change: Change that requires revising a limited number of directly and indirectly affected specification; and

Type 3—Global Change: Change so drastic that entire system gets affected.

This section presents an extension to the formalism for accommodating change in a system of PIL statements during or after the specification phase, without restarting the entire process. The extension is carried out by defining a database of PIL statements, their corresponding PG representation, and associations between the two representations. The database is created to keep track of input PIL statements and their corresponding graphical components in the PG representation. The unification and folding processes defined earlier merge the individual PGs, representing input statements, into a single PG by unifying nodes and folding edges. The processes, in turn, create new nodes and edges not present in any of the individual PGs. These newly created nodes/edges, and the corresponding inferred statements, need to be dealt carefully in the event of revising a PIL statement that may have contributed to these nodes/edges.

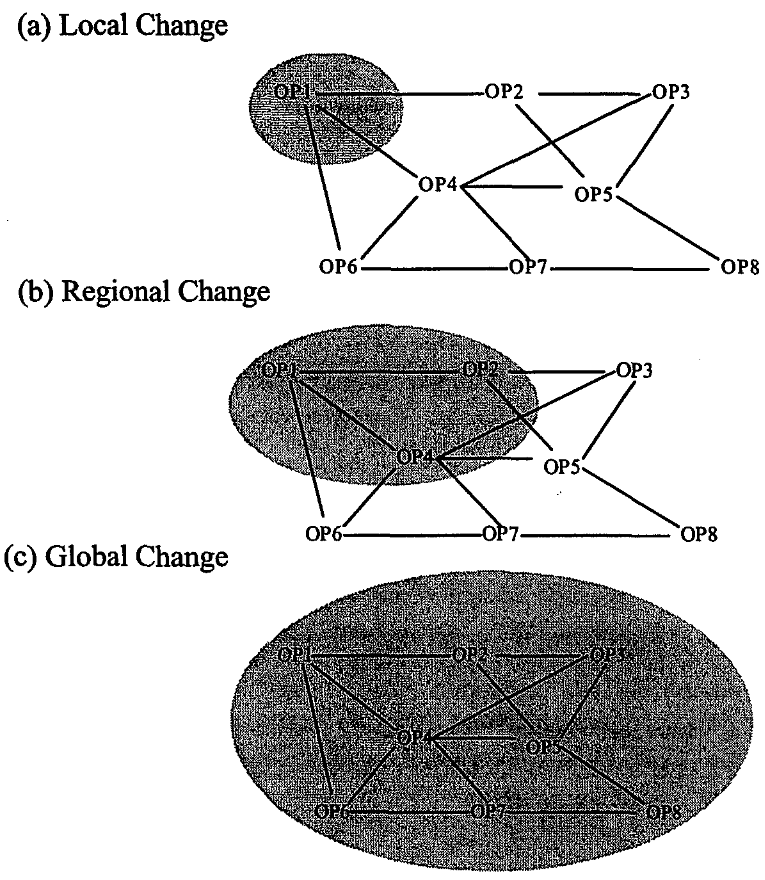


Figure 3.4.1. Types of Change

Definition 3.4.1: Database of PIL Statements Δ

A database of PIL statements is defined to be a 6-tuple Δ (S, V, C, ϕ, ν, δ), where

S: Set of PIL Statements

$S = \{s1, s2, s3, \ldots, sn\}$

V: Set of nodes in the corresponding PG representation (V, E, D, T)

C: Set of statements added to the database during unification and folding processes (Definitions 3.2.3 and 3.2.4)

ϕ: Mapping from PIL statements to their corresponding nodes in the PG

$(S \cup C) \rightarrow$ non-empty subsets of V

ν: Mapping from PG nodes to their corresponding PIL statements

$V \rightarrow$ non-empty subsets of $(S \cup C)$

δ: Mapping from statements in PIL to statements in C

$S \rightarrow$ subsets of C

The set C, in Definition 3.4.1, refers to the nodes/edges created as part of unification and folding processes and the corresponding inferred statements. A change in a statement contributing to one or some of these nodes/edges will have a direct impact on the inferred statements in the set C. The mappings ϕ, ν, and δ (Definition 3.4.1) keep track of this association between individual statements, both input and inferred, and the elements of the graph representing them. The database representation of a PIL system is updated every time a new statement is added, and/or unification and folding processes are applied to it. The following definitions describe the steps needed to update a database of PIL statements.

<u>Definition 3.4.2</u>: Adding a Statement to Δ

Let si be the statement to be added; the PG representing this statement is given as (Vi, Ei, Di, Ti); and the existing database is Δ (S, V, C, ϕ, ν, δ) with its PG representation given as (V, E, D, T). The new database and its PG, after the addition of si, are defined as follows:

Δ' (S', V', C', ϕ', ν', δ) PG' (V', E', D', T')

where

$S' = S \cup \{si\}$

$V' = V \cup Vi$

$C' = C$

$$\phi'(s) = \begin{cases} \phi(s) & \text{for } s \in S \\ Vi & \text{for } s = si \end{cases} \qquad E' = E \cup Ei$$

$$\nu'(v) = \begin{cases} \nu(v) & \text{for } v \in V \\ & \end{cases} \qquad T'(s) = \begin{cases} T(s) & \text{for } v \in V \\ Ti(s) & \text{for } v \in Vi \end{cases}$$

$$D'(e) = \begin{cases} \{si\} & \text{for } v \in Vi \\ D(e) & \text{for } e \in E \\ Di(e) & \text{for } e \in Ei \end{cases}$$

The addition of a statement to the database is followed by unification and folding operations that again change the database and the PG representation. The change to the database as part of the unification and folding operations is defined as follows. Definitions 3.4.3 and 3.4.4 describe the steps required in addition to steps already presented in Definitions 3.2.3, 3.2.5 and 3.2.6. For the sake of brevity, the contents of Definitions 3.2.3, 3.2.5 and 3.2.6 are not reproduced; the parts of Definitions 3.4.3 and 3.4.4 refer to the corresponding parts of Definitions 3.2.3, 3.2.5 and 3.2.6.

Definition 3.4.3: Unification with Δ

Let the database before unification is given as Δ (S, V, C, ϕ, ν, δ). The modified database after unification will be Δ' (S, V', C', ϕ', ν', δ'), where the components are redefined for each of the following cases.

(a) Refers to part (a) in Definition 3.2.3.

V' is constructed by the method presented in Definition 3.2.3.

C' = C

$$\phi'(s) = \begin{cases} [\phi(s) - \{vi, vj\}] \cup \{vi;vj\} & \text{for } s \in \nu(vi) \text{ or } \nu(vj) \\ \phi(s) & \text{otherwise} \end{cases}$$

$$\nu'(v) = \begin{cases} \nu(vi) \cup \nu(vj) & \text{for } v = vi;vj \\ \nu(v) & \text{for } v \in V' \end{cases}$$

(b) Refers to part (b) in Definition 3.2.3. Let c = Length [vi, vj] = (Stamp vj – Stamp vi)

V' = V

C' = C ∪ {c}

$$\phi'(s) = \begin{cases} \{vi, vj\} & \text{for } s = c \\ \phi(s) & \text{otherwise} \end{cases}$$

$$\nu'(v) = \begin{cases} \nu(v) \cup \{c\} & \text{for } v = vi \text{ or } vj \\ \nu(v) & \text{otherwise} \end{cases}$$

$$\delta'(s) = \begin{cases} \delta(s) \cup \{c\} & \text{for } s \in \nu(vi) \text{ or } \nu(vj) \\ \delta(s) & \text{otherwise} \end{cases}$$

Definition 3.4.4: Branch (Join) Folding with Δ

Let the database before folding is given as Δ (S, V, C, ϕ, ν, δ). The modified database after folding will be Δ' (S, V', C', ϕ', ν', δ'), where the components are redefined for each of the following cases.

(a) Refers to part (a) in Definition 3.2.5 (Definition 3.2.6.)

Let c = Length [vj, vk] = Length [vi, vk] – Length [vi, vj]

(c = Length [vj, vk] = Length [vk, vi] – Length [vj, vi] for Join Folding)

V' = V

$C' = C \cup \{c\}$

$$\phi'(s) = \begin{cases} \{vj, vk\} & \text{for } s = c \\ \phi(s) & \text{otherwise} \end{cases}$$

$$\nu'(v) = \begin{cases} \nu(v) \cup \{c\} & \text{for } v = vj \text{ or } vk \\ \nu(v) & \text{otherwise} \end{cases}$$

$$\delta'(s) = \begin{cases} \delta(s) \cup \{c\} & \text{for } s \in \nu(vj) \text{ or } \nu(vk) \\ \delta(s) & \text{otherwise} \end{cases}$$

(b) Refers to part (b) in Definition 3.2.5 (Definition 3.2.6). Let c = vj = vk

V' is constructed by the method presented in Definition 3.4.3.

$C' = C \cup \{c\}$

$$\phi'(s) = \begin{cases} [\phi(s) - \{vj, vk\}] \cup \{vj;vk\} & \text{for } s \in \nu(vj) \text{ or } \nu(vk) \\ \phi(s) & \text{otherwise} \end{cases}$$

$$\nu'(v) = \begin{cases} \nu(vj) \cup \nu(vk) \cup \{c\} & \text{for } v = vj;vk \\ \nu(v) & \text{for } v \in V' \end{cases}$$

$$\delta'(s) = \begin{cases} \delta(s) \cup \{c\} & \text{for } s \in \nu(vj) \text{ or } \nu(vk) \\ \delta(s) & \text{otherwise} \end{cases}$$

An implementation of the revision approach described in this section is presented in [Rauf and Zaidi, 2002] that uses a multi-layered PG structure to keep the database specifications in the lowest layer of a PG. The mappings between PG and the database representation are implemented with the help of linked-list data structures. The PIL statements (elements in S and

C) are kept as nodes in the lowest layer of the PG. The inference engine works at the higher layer to answer queries and infer new relations; however, a change in the inputs is processed at the lowest layer and its effects are propagated upwards. The effected parts of the PG determine the kind of change required to accommodate the change. Three types of revision operations are introduced in the PG approach by [Rauf and Zaidi, 2002]. ADD is the simplest; since it only requires a PG of the newly added statement to be incorporated in an existing processed PG of a system of PIL statements. The approach follows the construction in Definition 3.4.4 for a new PG, unifies and folds the new PG before making inferences. MODIFY can be considered as a DELETE followed by an ADD. The difficulty in deleting a PIL statement from a PG representation comes from the fact that unification and folding operations, while establishing new PIL relationships (i.e. elements in set C) among system intervals, also lose some information that is present in the input system. An abstract version of the DELETE operation, proposed by Rauf and Zaidi (2002) is given as follows. For a more technical description of the revision process, readers are referred to Rauf and Zaidi (2002).

Algorithm 3.4.1: DELETE

1. Let the database of PIL statements is given as $\Delta(S, V, C, \phi, \nu, \delta)$, the PG representing the system of PIL statements is PG (V, E, D, T), and the statement to be deleted is si, $si \in S$.
2. Identify the inferred statements in set C that are directly affected by si:

   $A = \{si\} \cup \delta(si)$

3. Identify the vertices of the PG that are affected by the statements in step (2):

   $P = \bigcup_{s \in A} \phi(s)$

4. Delete the elements of the set A from database Δ.
5. Identify the other PIL statements that also contribute to the vertices in step (3):

   $A_e = \bigcup_{v \in P} \nu(v)$

6. Identify the extended set of vertices in the PG that are affected by the change:

   $P_e = \bigcup_{s \in Ae} \phi(s)$

7. Remove each vertex v in P_e such that $\forall$ $\nu(v) \in A_e$. For other vertices in P_e set $\nu(v) = \nu(v) - A_e$. Reproduce the PG representation of the PIL statements in ($Ae \cap S$) on the leftover PG.
8. Reapply the unification (Definitions 3.2.3 and 3.4.3) and folding (Definitions 3.2.5, 3.2.6 and 3.4.4) to the resulting PG representation.
9. If DELETE is followed by an ADD, then the new statement is added before step 8 (Definition 3.4.2.)

The cardinality of the set P_e in step (6) of the algorithm 3.4.1, while compared with the cardinality of the set V, determines the extent of the change required by the revision.

Example 3.4.1

Consider the following set of PIL statements:

A, B, C, D Intervals

1: A < C

2: D o C

3: eB < eD

4: Stamp [sA] = 10

5: Stamp [sB] = 20

The database Δ (S, V, C, ϕ, ν, δ) after point graph is built for the above systems is as follows:

S = {1,2,3,4,5}

V = {sA,eA,sB,eB,sC,eC,sD,eD}

C = {I1}   I1: Length [sA, sB] = 10

ϕ = {(1,{sA,eA,sC,eC}),(2,{sD,eD,sC,eC}),(3,{eB,eD}),(4,{sA}),(5,{sB}) }

ν = {(sA,{1,4}),(eA,{1}),(sB,{5}),(eB,{3}),(sC,{1,2}),(eC,{1,2}),(sD,{2}),(eD,{2,3})}

δ = {(1,{}),(2,{}),(3,{}),(4,{I1}),(5,{I1})}

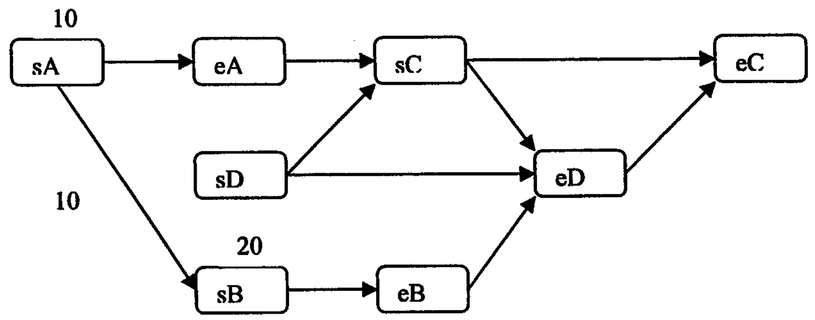


Figure 3.4.2. Point Graph for the given PIL Statements

Now suppose we want to delete the statement "Stamp [sB] = 20". The following is the step by step execution of delete method:

1. 5 is the statement to be deleted.
2. The inferred statement I1 depend upon 5 so A = {5, I1}.
3. Here P = {sB,sA}
4. At this point we remove entries corresponding to 5 and I1 from the database Δ and this is updated Δ:

   S = {1,2,3,4}

   V = {sA,eA,sB,eB,sC,eC,sD,eD}

   C = {}

   ϕ = {(1,{sA,eA,sC,eC}),(2,{sD,eD,sC,eC}),(3,{eB,eD}),(4,{sA})}

   ν = {(sA,{1,4}),(eA,{1}),(sB,{}),(eB,{3}),(sC,{1,2}),(eC,{1,2}),(sD,{2}),(eD,{2,3})}

   δ = {(1,{}),(2,{}),(3,{}),(4,{})}
5. Ae = {1,4}
6. Pe = {sB,sA,eA,sC,eC}
7. Update the database to reflect the removal of 1 and 4 so that the database Δ becomes:

   S = {1,2,3,4}

   V = {eB,sC,eC,sD,eD}

   C = {}

   ϕ = { (2,{sD,eD,sC,eC}),(3,{eB,eD})}

   ν = { (eB,{3}),(sC,{2}),(eC,{2}),(sD,{2}),(eD,{2,3})}

   δ = { (2,{}),(3,{}) }

At this point we add the statements 1 and 4 (Ae $\cap$ S) to the database. The point graph after step 7 is shown in the following figure. The shaded region shows the affected subgraph.

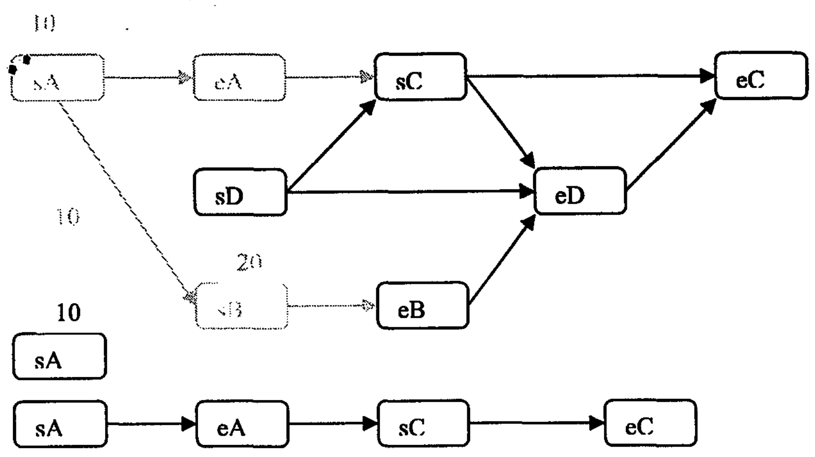


Figure 3.4.3. Point Graph after DELETE operation

8. Now perform unification and folding for the point graph.

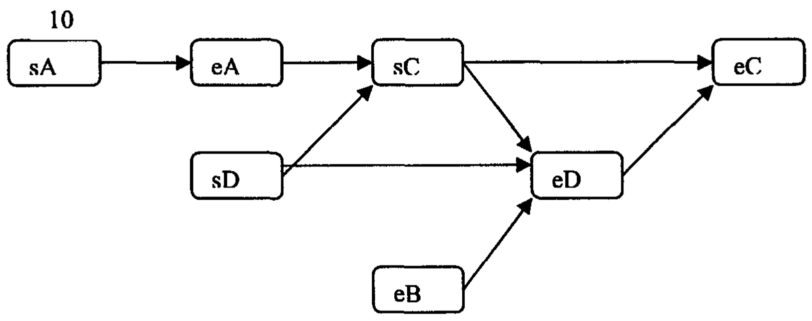


Figure 3.4.4. PG after unification and folding

3.5 Application to Temporal Systems

The growing need for a formal logic of time for modeling and analyzing real world systems has led to an emergence of various kinds of representations and reasoning schemes for temporal information. The earliest attempts at formalizing a time calculus date back to 1941 by Findlay, and 1955 by Prior. [Galton, 1987] Since then, there has been a number of attempts on issues related to this subject matter, like topology of time [Newton-Smith, 1980; Bochman, 1990a, 1990b; Galton, 1990; Van Benthem, 1991; Vila, 1994], first-order and modal approaches to time [Quine, 1965; Prior, 1967; Rescher and Urquart, 1971; Pnueli, 1977; McArthur, 1976; Chellas, 1980; Manna and Pnuelli, 1981; Reichgelt, 1989; Allen, 1983, 1984; Allen and Hayes, 1985; Allen and Ferguson, 1994], treatments of time for simulating action and language, etc. Several attempts on mechanizing the temporal reasoning processes have also been reported in the literature [Kahn and Gory, 1977; Allen and Koomen, 1983; Abadi and Manna, 1985; Cerro, 1985; Clarke et al., 1986; Dechter et al., 1991; Meiri, 1991; Keretho and Loganantharaj, 1991; Yao, 1994; Gerevini, 1995]. The list of references provided here is not at all exhaustive and may have missed some of the contributions. Some of the important sources for interested readers are Stock, 1997, Galton, 1987, Shoham, 1987, Gabbay et al., 1994, Anger et al., 1996, Barringer et al., 2000, TIME workshop series [Morris and Khatib, 1994, 1995, 1997, 1999; Chittaro et al., 1996; Goodwin and Trudel, 2000], the proceedings of the International Conference of Temporal Logic [Gabbay and Ohlbach, 1994; Gabbay and Barringer, 1997], and the proceeding of the Workshop on Spatial and Temporal Reasoning, 2003 [Guesgen et al., 2003]. Schwalb and Villa, 1998, Artale and Franconi, 2000, Augusto, 2001, and Ma and Knight, 2001, have presented surveys on some special classes of temporal reasoning research in their recent papers. The development of some of these formalisms has matured enough to attract comparative analyses for the computational aspects of these calculi and their subclasses [Golumbic and Shamir, 1992, 1993; Drakengren and Jonsson, 1997a, 1997b, 1997c ; Cervesato et al., 1997a, 1997b, 1998; Krokhin et al., 2003]. A number of researchers have attempted to use temporal reasoning formalisms for planning, plan merging, conditional planning, and planning with uncertainty problems [Allen and Koomen, 1983; McDermott, 1982; Allen et al., 1991; Allen, 1991a, 1991b, 1991c; Ma, 1999; Pollack and Horty, 1999; Onder and Pollack, 1999; Tsamardinos et al. 2000]. Other applications of temporal logics include specification and verification of real-time, reactive planners [Rosenchein and Kaebling, 1995; Williams and Nayak, 1996], and specification of temporal-extended goals and search control rules [Bacchus and Kabanza, 1996]. An earlier introduction to some of the literature on spatial reasoning can be found in Davis (1986, 1990), Russell and Norvig (1995), and Stock (1997). Some of the more recent work and related references can be found in Guesgen et al., (2003).

Table 3.5.1 Temporal Equivalents of PIL Relations

PIL Relation/Function Name	**Corresponding Temporal Equivalent**
Atomic Relations	
<	Before
M	Meets
O	Overlaps
S	Starts
D	During
F	Finishes
=	Equals
Stamp	Time
Length	Length
Some Compound Relations	
<= (for points only)	
<s (for point and interval)	
<m (for intervals only)	Precedes
Osd	Ends_During
$o^{-1}df$	Starts_During
$ss^{-1}=$	Starts_With
$ff^{-1}=$	Ends_With
$<mod^{-1}f^{-1}$	Starts_Before
<mo	Starts_Before_Starts
<mosd	Ends_Before_Ends
$<moo^{-1}sdd^{-1}ff^{-1}=$	Starts_Before_Ends

This section presents an application of PIL for modeling temporal situations. Table 3.5.1 lists the PIL relations, function names, and their corresponding temporal lexicon. The table also suggests some high-level temporal relations that can be used to represent compound PIL relations. The task of designing a comprehensive and suitable language for temporal relations is left as a choice for the user, who may define his/her own (natural language) constructs for the entire set of compound relations given in Tables 3.1.3-5. A system of temporal statements can, therefore, be constructed using the temporal lexicon with the PIL syntax. The temporal version of the logic is termed Point Interval Temporal Logic (PITL). Once the temporal inputs are specified using the new lexicon, the rest of the formalism is identical to the approach presented in the previous sections. It is obvious to conclude that the notions of interpretation, satisfiability, inconsistency, and inference in PIL are equivalent to the corresponding temporal interpretation, temporal satisfiability, temporal inconsistency, and temporal inference in PITL. In this section, we take the temporal implementation of PIL a step further by introducing a suite of PG-based temporal analyses for a possible application to mission planning problems.

Mission Planning

This section presents a subclass of PITL (Definition 3.5.1) for modeling temporal requirements and/or constraints of a mission to be planned. The points and intervals of the logic correspond to time stamps and time delays, respectively, associated with events/activities in the mission as constraints to or as resultants of a planning process. The lexicon of the logic offers the flexibility of both qualitative and quantitative descriptions of temporal relationships between points and intervals of the system. The definition of the subclass (Definition 3.5.1), however, puts some restrictions on the type of temporal information that can (or cannot) be handled by the analysis presented in this section. (Note that the restrictions in Definition 3.5.1 do not apply to the approach presented in the previous sections; a generic system of PITL statements can be processed by the methods presented and the inference mechanism of PITL can be invoked to infer unknown temporal relationships between system intervals.)

Definition 3.5.1: Subclass of PITL, $\mathcal{A}$

The subclass $\mathcal{A}$ of PIL is described with the help of the following requirement on a system of PIL statements Δ:

A system of PITL statements $\Delta \in \mathcal{A}$ if in the string representation of statements in Δ, every strict inequality (< or >) between two points p1 and p2 is accompanied by a length expression for the distance between the two points. The following is a set of necessary conditions for the PIL statements that follow this characterization.

1. All intervals defined in the system are provided with their lengths, i.e., $\forall$ X, where X = [sX, eX], 'Length X = d' $\in \Delta$, for some d $\in \Re$;
2. for a pair of points, X and Y, if 'X Before Y' $\in \Delta$, then 'Length[X, Y] = d' $\in \Delta$, for some d $\in \Re$;
3. for a point X and an interval Y, if 'X Before Y' $\in \Delta$, then 'Length[X, sY] = d' $\in \Delta$, for some d $\in \Re$;
4. for a point X and an interval Y, if 'Y Before X' $\in \Delta$, then 'Length[eY, X] = d' $\in \Delta$, for some d $\in \Re$;
5. for a pair of intervals, X and Y, if 'X Before Y', 'X Overlaps Y', or 'X During Y' $\in \Delta$, then: 'Length[sX, sY] = d', 'Length[sX, eY] = d', 'Length[sY, eX] = d', or 'Length[eX, eY] = d' $\in \Delta$, for some d $\in \Re$;

Corollary 3.5.1: PG Representation of the Subclass

The unified and folded PG representing a system in $\mathcal{A}$ has a *total* edge-length function.

The approach presented in this section requires the temporal constraints of a mission to be converted to PITL statements. The system of PITL statements should conform to the definition of the subclass $\mathcal{A}$. The temporal system is then converted to its PG representation. The PG, so obtained, is processed by applying unification and folding processes (Definitions 3.2.3, 3.2.5, 3.2.6). The folded PG is checked for inconsistency by the approach presented in Section 3.3. The verification of PG either reports infeasible temporal requirements in the input, or ensures the fact that the input PITL system is satisfiable. The inference mechanism of the logic can now be invoked, for a consistent PITL system, to determine temporal relations between intervals/points of interest. In order to construct a model (Definition 3.3.4) of the temporal system, the PG is added with a pair of *source* and *sink* nodes (Definition 3.5.2). At this point, an *optimized* model of the PITL system can be constructed by solving the mathematical program defined in Definition 3.5.3 for the PG representation. The model is termed optimized for the reason that it constructs an interpretation of the system with the minimized start-to-end ($V_{out} - V_{in}$) time duration. Alternatively, a graph-based analysis can be used to construct a similar model of the temporal system. The time stamps on individual nodes are not considered in the two approaches;

the stamps can be ignored without any loss of generality. The time stamp can be easily incorporated either before or after the analysis that follows. Once a plan is constructed using the approach, the plan can be shifted on a timeline to match with the stamps provided in the input PITL statements. •

Definition 3.5.2: Source and Sink Nodes to PG

A source node V_{in} and a sink V_{out} node are added to the PG representation of a system of PITL statements by applying the following:

(a) $\forall vi, vi \in V$ such that $*v = \phi$ (i.e., null set), connect the source node V_{in} to all vi's by LE type edges (V_{in}, vi);

(b) $\forall vi, vi \in V$ such that $v^* = \phi$, connect the sink node V_{out} to all vi's by LE type edges (vi, V_{out}).

Definition 3.5.3: Mathematical Program Representing PG

Given a PG ($V \cup \{V_{in}, V_{out}\}$, E_A, D, T), where $E_A = E \cup E_{\leq}$, a mathematical program for constructing an interpretation of the PITL represented by PG is defined as:

Objective Function:

$$\text{Minimize } V_{out} - V_{in}$$

Subject to:

$$vj - vi = D(vi, vj), \qquad \forall\ (vi, vj) \in E$$

$$vi \leq vj, \qquad \forall\ (vi, vj) \in E_{\leq}$$

$$vi \geq 0, \qquad \forall\ vi \in V$$

The graph-based approach assigns three parameters to each node in the PG representation. The parameter values are calculated by running an analysis on the graph. The values of these parameters help determine the *critical activities* (Definition 3.5.9, below) and time floats/slacks (Definition 3.5.10, below) for intervals in the system, and *interval/point activities* (Definitions 3.5.7-8) defined for the PG under consideration. The three parameters are termed as *earliest occurrence (Ev), late occurrence (Lv), and latest occurrence (Tv)* of a node 'v', and are formally defined in Definitions 3.5.4-6. The analysis applies two passes through the PG representation. The first, forward pass (Definition 3.5.4), calculates the value for the earliest occurrence time of a node; the other, reverse pass (Definitions 3.5.5-6), calculates the values for the late and latest occurrences of a node in the PG. Figures 3.5.1-2 illustrates the two passes with the help of example cases.

Definition 3.5.4: Earliest Occurrence of a Node, Ev, in PG – Forward Pass

The earliest occurrence Ev of a node v, v ∈ V, is defined to be the smallest time stamp on the node that satisfies the earliest occurrences of the preceding nodes, i.e.,

Let *v = {vi}

$$Ev = \begin{cases} Evi + D(vi, v), & \text{for } (vi, v) \in E \text{ and } |{*v}| = 1 \\ \max\limits_{i} [Evi], & \forall (vi, v) \in E_{\leq} \\ \max\limits_{i} [Evi, Evk + D(vk, v)], & \text{for } (vk, v) \in E \\ 0, & \text{otherwise} \end{cases}$$

For a *non-critical* interval/activity [v1, v2] (Definitions 6.7-9), Ev1 represents the *earliest start time* of the activity.

Definition 3.5.5: Late Occurrence of a Node, Lv, in PG – Reverse Pass I

The late occurrence Lv of a node v, v ∈ V, is defined to be the largest time stamp on the node that satisfies the earliest occurrences of the following nodes, i.e.,

Let v* = {vi}

$$Lv = \begin{cases} Lvi - D(v, vi), & \text{for } (v, vi) \in E \text{ and } |v^*| = 1 \\ \min\limits_{i} [Evi], & \forall (v, vi) \in E_{\leq} \\ \min\limits_{i} [Evi, Lvk - D(v, vk)], & \text{for } (v, vk) \in E \\ Ev, & \text{otherwise} \end{cases}$$

Definition 3.5.6: Latest Occurrence of a Node, Tv, in PG – Reverse Pass II

The latest occurrence Tv of a node v, v ∈ V, is defined to be the largest time stamp on the node that satisfies the latest occurrences of the following nodes, i.e.,

Let v* = {vi}

$$Tv = \begin{cases} Tvi - D(v, vi), & \text{for } (v, vi) \in E \text{ and } |v^*| = 1 \\ \min\limits_{i} [Tvi], & \forall (v, vi) \in E_{\leq} \\ \min\limits_{i} [Tvi, Tvk - D(v, vk)], & \text{for } (v, vk) \in E \\ Ev, & \text{otherwise} \end{cases}$$

For a *non-critical* interval/activity [v1, v2] (Definitions 3.5.7-9), Tv2 represents the *latest completion time* of the activity.

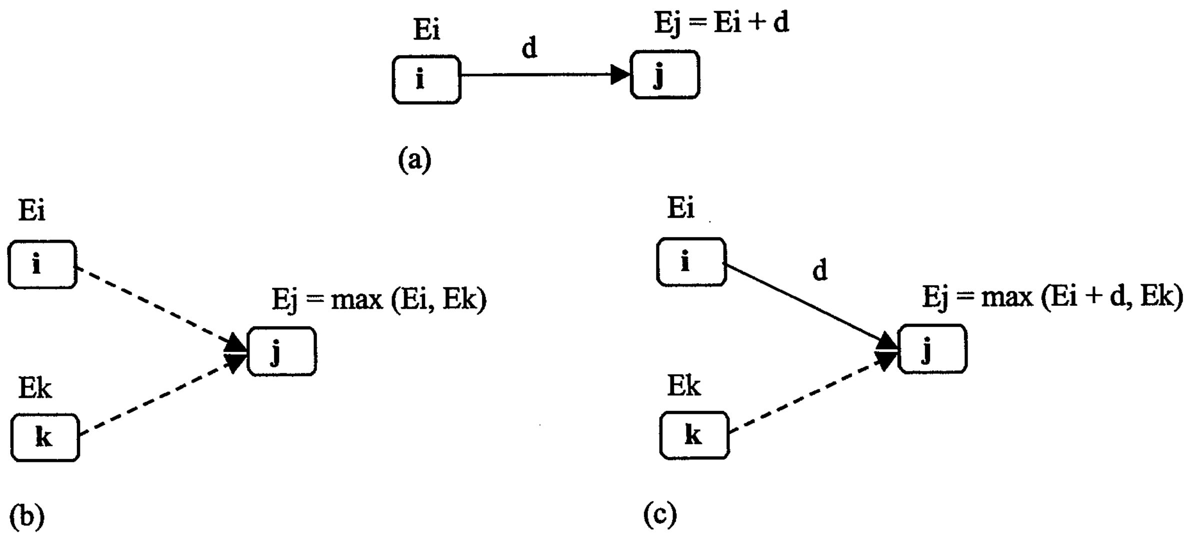


Figure 3.5.1. Illustration of Forward Pass

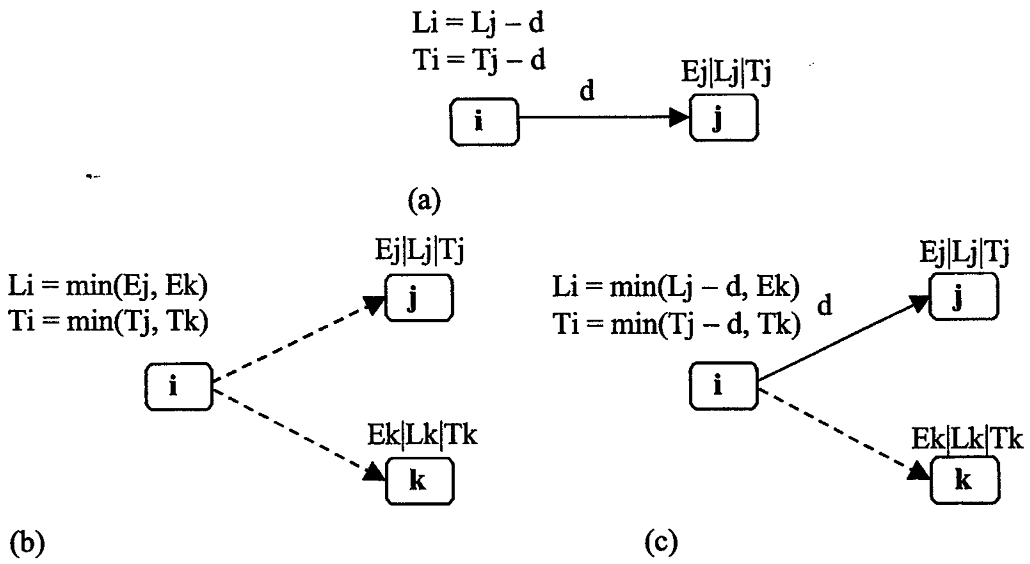


Figure 3.5.2. Illustration of Reverse Pass

Definition 3.5.7: Point Activity

A node v ∈ V is called a point activity. A point, start and end points of an interval, in the PITL system are all point activities in the PG representation of the PITL system.

Definition 3.5.8: Interval Activity

An interval [v1, v2], where v1, v2 $\in$ V, is called an interval activity if the two time points represented by the nodes v1 and v2 are the two end points of a path comprising of LT type edges only.

Note that the definition of interval activities, in Definition 3.5.8, extends the notion of intervals in a PITL system by including composite and parts of PITL intervals to be defined as interval activities in a PG representation.

Definition 3.5.9: Critical Activity

An activity is defined to be critical if:

(a) a delay in its start will cause a delay in the completion time of the entire mission, i.e.,

(i) for a point activity v $\in$ V, Ev = Tv;

(ii) for an interval activity [v1, v2], where v1, v2 $\in$ V, v $\in$ [v1, v2], Ev = Tv.

or

(b) for an interval activity, it 'Meets' or is met by (Meets^{-1}) another critical activity; for a point activity, it 'Starts' and/or 'Ends' another critical activity.

or

(c) an earliest (or latest) occurrence of its start node does not ensure an earliest (or latest) occurrence of its end node, i.e.,

for [v1, v2], $Ev1 + D([v1, v2]) < Ev2$, or

$Tv1 + D([v1, v2]) < Tv2$

Definition 3.5.10: Total Float (TF) and Free Float (FF)

Total Float (TF) is the difference between the maximum time available to perform an activity and its duration. Free Float (FF) is defined by assuming that all the activities start as early as possible. It is the excess time available over its duration.

(a) Total float (TF) and free float (FF) for a non-critical point activity, v, is calculated as:

$TFv = Tv - Ev$

$FFv = Lv - Ev$

(b) Total float (TF) and free float (FF) for a non-critical interval activity, [v1, v2], is calculated as:

$$\begin{aligned} TF_{[v1, v2]} &= Tv2 - Ev2 \\ &= Tv1 - Ev1 \end{aligned}$$

$$FF_{[v1, v2]} = Lv2 - Ev2$$
$$= Lv1 - Ev1$$

(For all critical activities TF = FF = 0)

The condition (c) in Definition 3.5.9 presents an interesting and a new notion of critical activities in the context of planning and scheduling literature. The condition represents an activity that, for a given start-to-end mission duration, is required to start and end at specific times, in order to satisfy the preceding and following activities timings. But, the difference between the two times, start and end, is greater than the actual duration of this activity. This difference between the actual duration and the required duration is called *stretch float (SF)*. Example 3.5.1 illustrates the concept with the help of an example.

Definition 3.5.11: Stretch Float (SF)

For a critical activity [v1, v2] of type defined in Definition 3.5.9c, Stretch Float (SF) is defined to be the excess time available over the duration between the earliest occurrences of its start 'v1' and end 'v2' nodes, i.e.,

$$SF_{[v1, v2]} = Ev2 - Ev1 - D([v1, v2]) \quad [\text{or } Tv2 - Tv1 - D(v1, v2)]$$

The stretch float, if exists, presents the following set of alternates to a mission planner.

(a) For a critical activity [v1, v2] with SF, if any one of the following holds:

i. $Lv1 + D([v1, v2]) = Ev2$;

ii. $Tv1 + D([v1, v2]) = Lv2$;

iii. $Tv1 + D([v1, v2]) = Ev2$

Then, the activity is scheduled in the corresponding interval.

(b) For the activity, $Tv1 + D([v1, v2]) < Ev2$ – the activity if started at the latest time still ends earlier than required by some of the preceding activities, but the activity's end time can be delayed (stretched) by an amount equal to its SF after its start. Then, the activity is stretched.

(c) For an activity that does not satisfy any conditions in part (a) and cannot be stretched—part (b) —, the mission cannot be planned without extending the start-to-end duration of the mission. (See Example 3.5.1, Figure 3.5.4.)

Example 3.5.1

Let a mission's requirements be specified in terms of the following system of PITL statements:

Δ:	A, B, C, D, E intervals		
	Length A = 5	Length B = 5	Length C = 5
	Length D = 2	Length E = 10	A Meets B
	C Meets D	C Precedes B	eE Precedes eD

The approach takes the statements in Δ and converts them to their corresponding PG representation. The PG is unified, folded, and verified for satisfiability. A pair of source and sink nodes is added to the PG, and forward and reverse passes are applied to the resulting PG. Figure 3.5.3 shows the PG with all the parameters values calculated for each node in the graph. The start-to-end ($V_{out} - V_{in}$) delay of 10 time units is the shortest possible duration for the mission to be accomplished, provided none of the constraints is violated. An inspection of the PG reveals the fact that all the activities involved are critical. The activity D also satisfies the condition in Definition 3.5.9c, with an SF value of 3 time units. The SF suggests that the activity needs to be stretched from a duration of 2 time units to 5 time units, should the mission need to be accomplished with the minimal 10 time units with all requirements met. If, on the other hand, the activity D cannot be stretched, the mission duration needs to be extended by an amount equal to SF_D. Figure 3.5.4 presents the situation where a dummy activity 'F' is added to the system with 'Length F = 13' before re-calculating the parameters values. The dummy activity is added to force the start-to-end time to be at least equal to its length of 13 time units. All the activities in Figure 3.5.4 are critical, with their feasible time stamps underlined. The feasible time stamps are selected by first looking at the critical activities and the feasible stamps on the nodes involved. In the example case, the critical activity D can only start at '$T_{sD} = 8$' which, in turn, makes the start of another critical activity C to be '$T_{sC} = 3$'. Similarly, the feasible time stamps of other critical activities, i.e., A, B, and E, are selected. The values in Figure 3.5.4, therefore, show the *only* feasible schedule for the activities involved for the mission duration of 13 time units.

Finally, the PG corresponding to a mission's requirements, with the values of the parameters calculated, can be used to construct a time chart, e.g., Gant chart, showing the start and finish times for each activity as well as its relationship to other activities. It also must pinpoint the critical activities. For non-critical activities the plan also must show the amount of slack or float times that can be used advantageously when such activities are delayed or when limited resources are to be used. The PG representation and the time chart can, therefore, be used for a real-time and periodic control of the plan. The PG may be updated and analyzed, and, if necessary, a new plan/schedule is determined for the remaining portion of the mission in a dynamic environment. An extension to the formalism that proposes an improved graph-based approach to calculate the parameter values by employing a combination of the two (forward and reverse) passes will be presented in a forthcoming paper.

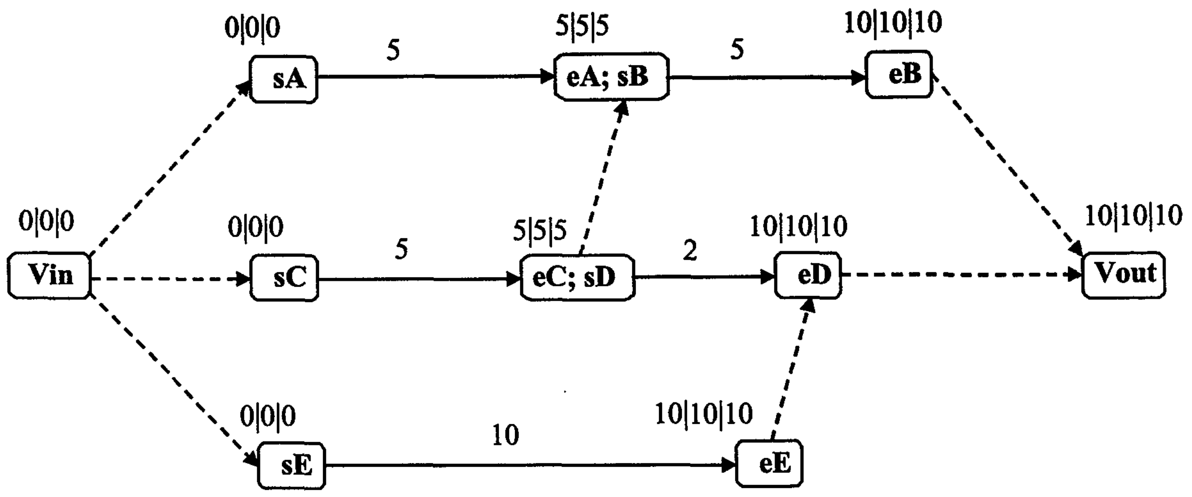


Figure 3.5.3. PG for Example 3.5.1

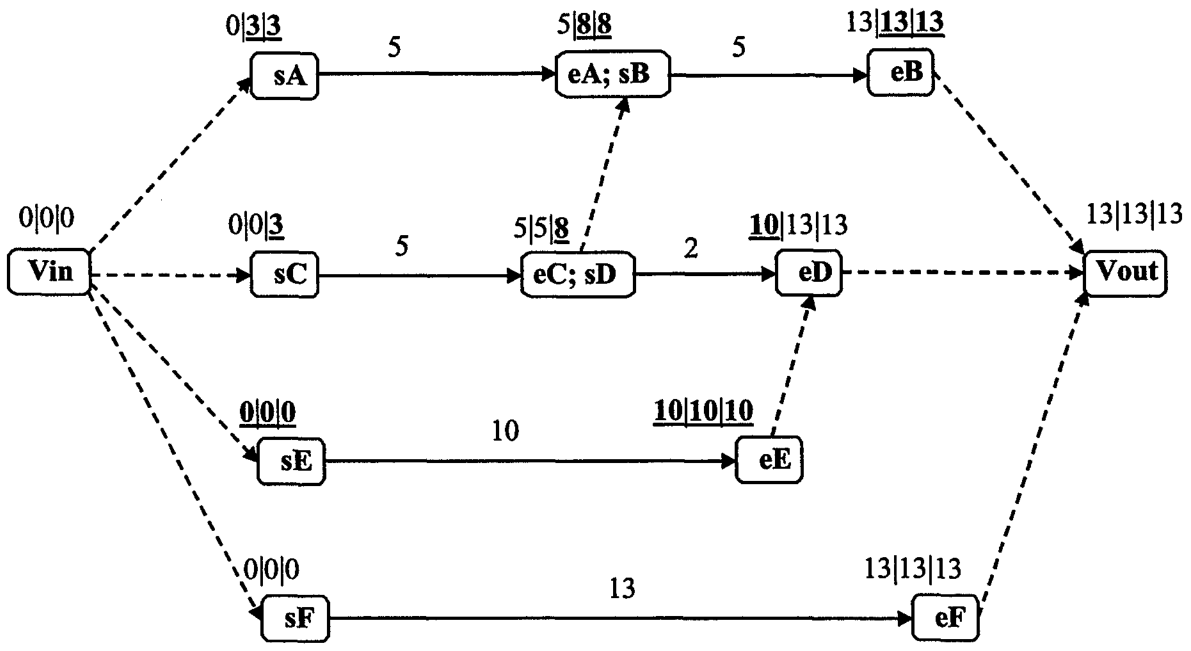


Figure 3.5.4. Modified PG for Example 3.5.1

Application

This section provides a fictitious but real world example to illustrate how some of the features of the PIL and PITL could be applied to military planning and execution problems. The illustration is for a precision engagement against a Time Critical Target (TCT). A scenario is presented in which several assets must concurrently perform activities with implicit synchronization in order to attack a target of importance. The target is time critical in that it is difficult to locate and when it is located, it must be struck in a very short time, otherwise it will disappear.

Assume the following facts and constraints apply to the planning for precision engagement of TCTs. There is a list of high value TCTs that when located and identified need to be attacked quickly with precision engagement weapons. When such a target is found, a weapon platform such as an attack aircraft must ingress to a weapon launch point to release a precision-guided weapon (PGW). During the ingress, the on-board navigation and guidance processor of the PGW will be uploaded with the precise data it needs to fly to and hit the target. During the ingress and PGW update activities, a local, on site, aid to the navigation and guidance activity must participate in providing updates to the PGW. This local, on site activity must cease just prior to the weapon striking the target. Once the weapon is launched, the launch platform egresses the area.

The plan for this scenario can be directly mapped to Example 3.5.1, Section 3.5, mission requirements as shown in Table 3.5.2. The table shows the five activities together with the PITL statements representing the mission operational concept. The additional constraints are described in Table 3.5.3 with their corresponding PITL statements.

Table 3.5.2. Mission Requirements

Interval ID	**Activity Description**	**Corresponding PITL Statement**
A	Weapon Platform ingresses to PGW launch point	Length A = 5
B	Weapon Platform egresses from PGW launch point	Length B = 5
C	Target parameters are uploaded into the PGW navigation processor	Length C = 5
D	PGW is launched and flies to the Target	Length D = 2
E	Local, on site activity provides navigation and guidance update to PGW	Length E = 10

Table 3.5.3. Additional Constraints

Natural Language Description	Corresponding PITL Statement
The platform will not loiter in the area due to threat considerations	A meets B
The PGW is launched immediately after the target parameters are uploaded	C meets D
The PGM launch precedes the egress	C Precedes B
Local, on site activity must cease just prior to weapon striking the target	eE Precede eD

These activity descriptions and constraints equate to the mission requirement PITL statements of Example 3.5.1. Following the approach presented, these mission requirements are converted to the corresponding PG representation as shown in Figure 3.5.3. From this figure, the minimum time required to execute the mission is 10 time units (perhaps 10 minutes). The analysis also reveals that there is a stretch float (SF) condition associated with activity interval D, the PGW fly out activity. Further review of the scenario indicates that this activity cannot be stretched because the fly out time is fixed. Thus, the second PG shown in Figure 3.5.4 is created. This PG shows that the total mission time is 13 time units. Furthermore, the start and end times of all activities are captured in the PG. Thus the local, on site activity starts at time 0, the Ingress and the PGW upload start at time 3. The PGM launch occurs at time 8 and commences the Egress activity. The PGW strikes the target at time 13 just after the local, on site activity ceases. This plan provides a total mission view that can be used to provide to the individual resources that are carrying out the plan the critical start and complete times for their activities to ensure the implicit synchronization of the concurrent activities is accomplished.

This illustration is but a simple vignette. It is included to demonstrate one of many potential real-world applications of the approach presented in this report. It is believed that the approach is capable of providing very powerful analytical capabilities support both real world deliberate and near real time planning and plan repair problems.

Temporal Analysis of Timed Influence Nets

The modeling of the causal relationships in Timed Influence Nets (TINs) is accomplished by creating a series of cause and effect relationships between some desired effects and the set of actions that might impact their occurrence in the form of an acyclic graph. The actionable events in a TIN are drawn as root nodes (nodes without incoming edges). A desired effect, or an objective in which a decision maker is interested, is modeled as a leaf node (node without outgoing edges). Typically, the root nodes are drawn as rectangles while the non-root nodes are drawn as rounded rectangles. Figure 3.5.5 shows a partially specified TIN. Nodes B and E represent the actionable events (root nodes) while node C represents the objective node (leaf node). The directed edge with an arrowhead between two nodes shows the parent node promoting the chances of a child node being true, while the roundhead edge shows the parent node inhibiting the chances of a child node being true. The inscription associated with each arc shows the corresponding time delay it takes for a parent node to influence a child node. For instance, event B, in Figure 3.5.5, influences the occurrence of event A after 5 time units.

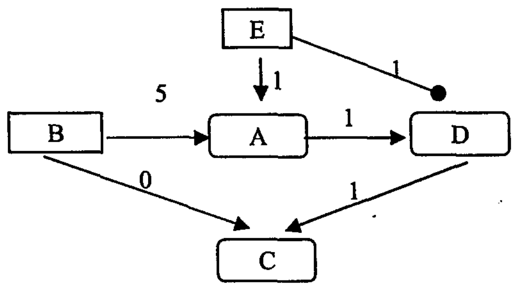


Figure 3.5.5. An Example Timed Influence Net (TIN)

The purpose of building a TIN is to evaluate and compare the performance of alternative courses of actions. The impact of a selected course of action on the desired effect is analyzed with the help of a *probability profile*. Consider the TIN shown in Figure 3.5.5. Suppose the following *input scenario* is decided: actions B and E are taken at times 1 and 7, respectively. Because of the propagation delay associated with each arc, the influences of these actions impact event C over a period of time. As a result, the probability of C changes at different time instants. A probability profile draws these probabilities against the corresponding time line. The probability profile of event C is shown in Figure 3.5.6.

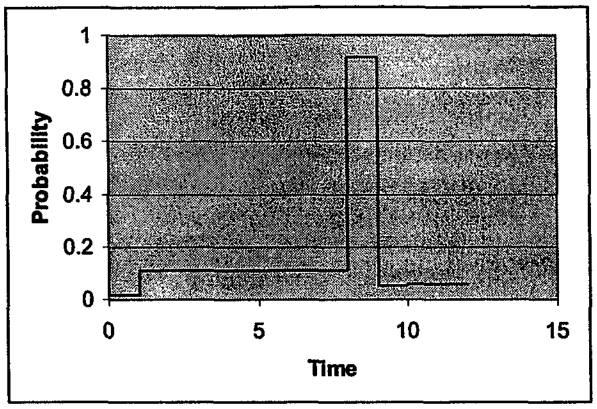


Figure 3.5.6. Probability Profile for Node C

The following items characterize a TIN:

1. A set of random variables that makes up the nodes of a TIN. All the variables in the TIN have binary states.
2. A set of directed links that connect pairs of nodes.
3. Each link has associated with it a pair of parameters that shows the causal strength of the link (usually denoted as g and h values).
4. Each non-root node has an associated baseline probability, while a prior probability is associated with each root node.
5. Each link has a corresponding delay d (where $d \geq 0$) that represents the communication delay.
6. Each node has a corresponding delay e (where $e \geq 0$) that represents the information processing delay.
7. A pair (p, t) for each root node, where p is a list of real numbers representing probability values. For each probability value, a corresponding time interval is defined in t. In general, (p, t) is defined as

   $([p_1, p_2, \ldots, p_n], [[t_{11}, t_{12}], [t_{21}, t_{22}], \ldots., [t_{n1}, t_{n2}]])$,

   where $t_{i1} < t_{i2}$ and $t_{ij} > 0 \ \forall \ i = 1, 2, \ldots., n$ and $j = 1, 2$

The last item in the above list is referred to as input scenario, or sometimes (informally) as course of action. Formally, a TIN is described by the following definition.

Definition 3.5.12 Timed Influence Net (TIN)

A TIN is a tuple $(V, E, C, B, D_E, D_V, A)$ where

V: set of Nodes,

E: set of Edges,

C represents causal strengths:

$E \rightarrow \{ (h, g) \text{ such that } -1 < h, g < 1 \}$,

B represents Baseline / Prior probability: $V \rightarrow [0,1]$,

D_E represents Delays on Edges: $E \rightarrow N$,

D_V represents Delays on Nodes: $V \rightarrow N$, and

A (input scenario) represents the probabilities associated with the state of actions and the time associated with them.

$A: R \rightarrow \{([p_1, p_2, \ldots, p_n],[[t_{11},t_{12}], [t_{21},t_{22}], \ldots,[t_{n1},t_{n2}]])$

such that $p_i = [0, 1]$, $t_{ij} \rightarrow Z$ and $t_{i1} \leq t_{i2}$,

$\forall\, i = 1, 2, \ldots, n$ and $j = 1, 2$ where $R \subset V \}$

This section explains how the integration of TIN and PG formalisms adds new suit of techniques for analyzing complex uncertain situations [Haider et al., 2005]. The proposed techniques aid a system modeler in gaining a better insight of the impact of a selected course of action on desired effect(s). The PG representation of a corresponding TIN answers queries regarding certain temporal characteristics of an effect's probability profile. Furthermore, the PG aids a system modeler by explaining what needs to be done in order to achieve a certain effect at a specific time instant. If the requirements for achieving effects at certain time instants are not temporally consistent, then the PG helps in understanding the reasons for inconsistencies. Both types of temporal analyses assume that a corresponding PG has been constructed from a TIN.

Creating a Point Graph from a Timed Influence Net

The steps involved in generating a PG from a corresponding TIN are presented in Table 3.5.4. The following is the illustration of the conversion approach with the help of the sample TIN of Figure 3.5.5. For the example case: R = {B, E} and F = {C}.

Step 1: In this step, all the paths from the root nodes to the leaf nodes are determined. For instance, there are four distinct paths in the TIN of Figure 3.5.5:

(i) B – A – D – C (ii) B – C

(iii) E – A – D – C (iv) E – D – C

Step 2: This step transforms each path into a corresponding PG. A unique subscript is added to all the nodes in the path during this transformation. Thus, path B-A-D becomes B1-A1-D1; path B becomes A2-D2-C2; and so on. The delays attached with each arc in the TIN are transformed to length expressions in PG. The corresponding PIL statements are given below:

Length[B1,A1] = 5	Length[A1,D1] = 1
Length[D1,C1] = 1	Length[E3,A3] = 1
Length[A3,D3] = 1	Length[D3,C3] = 1
Length[E4,D4] = 1	Length[D4,C4] = 1

B2 Equals C2

It should be noted that the time delay between B and C is 0 time unit. Thus, both events represent the same temporal point. The PGs obtained as a result of Step 2 are shown in Figure 3.5.7.

Table 3.5.4. A PG Construction from a TIN

```
Given a TIN
    R: Set of Root Nodes (Actionable Events)
    F: Set of Leaf Nodes (Desired Effects)
1. For each r ∈ R find all the paths leading to a f ∈ F.
   Apply this step for all f ∈ F.
2. Add a unique subscript to each node in an individual
   path obtained in Step 1.
3. Represent each path as a PG where a node in the path
     becomes a vertex and a delay d (d >0) on an arc
   between two vertices v1, v2 becomes Length(v1,v2)=d
   in the PG.
4. For each set of vertices in PG that represent a root
    node in TIN, add a temporal equality constraint
    'Equal' among its elements.
The following step is executed once an input scenario is
provided.
5. Based on the input scenario, assign time stamps to
   vertices representing root nodes .
6. Construct an aggregate PG using temporal statements
    provided in Steps 3-5 after applying the unification
    and folding operations.
```

Step 3: This step adds temporal relation 'Equals' between points that represent a particular root node in the corresponding TIN. Since R = {B, E}, the following information is provided to the PIL engine:

B1 Equals B2

E3 Equals E4

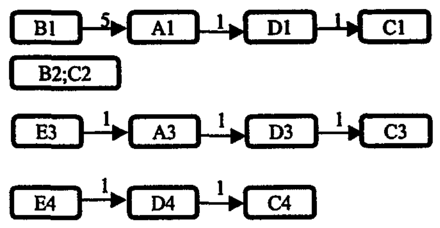


Figure 3.5.7. PGs Corresponding to Paths in the TIN

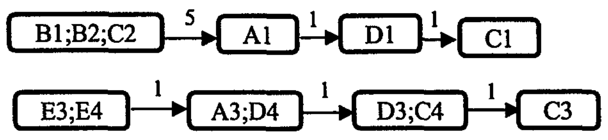


Figure 3.5.8. Branch Folded PGs

Based on the given information, unification and folding operations are applied on the PGs of Figure 3.5.7. The resulting PGs are shown in Figure 3.5.8.

Step 4: Let an input scenario be given: suppose B occurs at time 1, while E occurs at 7. This information is added to the set of temporal statements described in the earlier steps. Thus, the following statements are added:

Stamp[B1] = 1

Stamp[E3] = 7

This last set of information results in further unification and folding of the PGs of Figure 3.5.8. The final consolidated PG is shown in Figure 3.5.9.

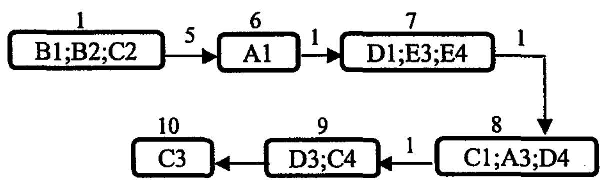


Figure 3.5.9. PG with Input Scenario

Temporal Queries

Once a PG is obtained from a TIN, it can then be used to explain certain temporal characteristics of a probability profile. Consider the profile shown in Figure 3.5.6. Suppose a system modeler is interested in knowing what causes a change in the probability of event C at time 8. The algorithm that answer this and similar queries is presented in Table 3.5.5 and is explained below, with the help of an example.

Table 3.5.5. Answering Temporal Queries using a PG

```
Given a PG, a TIN, v: variable of interest,
      t: time of interest, C: list of Causes
1. Initialize C to null.
2. Determine the subscripts of v at time t. Let
   S = [s1, s2, ..., sn] be the list of subscripts.
3. For each element s in S:
     (i) Starting from the root of the PG, search the PG
         until the first variable with the subscript s is
         identified. Let x be such a variable.
     (ii) Let m be the time stamp associated with x.
     (ii) Add (x, m) to C.
4. Report the list C.
```

The algorithm first identifies the subscript(s) of the variable of interest for a given time stamp. For instance, the subscript of C at time 8 is '1'. Starting from the root of the PG, the algorithm, in the next step, searches the graph for the subscript until it finds the first variable (or set of variables if they share the subscript) that matches the subscript. For instance, in the PG of Figure 3.5.9, the first element having subscript '1' is B. The time stamp associated with B1 is 1. Thus, a change in the profile of C at time 8 is because of action B occurring at time 1. If more than one action cause a change in the probability of C at time 8, then all of them are reported along with the time of their occurrences. Furthermore, if multiple paths exist between an action node and a desired effect, in a TIN, the algorithm can identify the path through which the action has impacted the effect node. For the example under consideration, the path through which B impacted C at time 8 is: B – A – D – C.

What-If Analysis

The PG obtained can also aid in performing what-if analyses. Suppose after observing the probability profile of Figure 3.5.6, the system modeler is interested in knowing what needs to be done, in order to combine the impacts that reach C at times 8 and 9. The algorithm that accomplishes this task is presented in Table 3.5.6.

The algorithm assumes that a PG based on an input scenario has already been constructed (Figure 3.5.7 in the current context). As stated above, the modeler is interested in combining the impacts that reach node C at time 8 and 9; thus, list V has elements [C, C], while list T has elements [8, 9]. The first four steps of the algorithm are the same as the algorithm given in Table 3.5.4, and therefore, are not explained below. In Step 5, the subscripts of elements in V at corresponding times, described in T, are determined. Thus, C at time 8 has subscript '1', while C at time 9 has subscript '4'. As a result of this step, list S consists of [C1, C4]. Step 6 adds the following statement: C1 Equals C4

Table 3.5.6. What-If Analysis Using a PG

```
Given a TIN, a PG G1
    R: Set of Root Nodes (Actionable Events)
    F: Set of Leaf Nodes (Desired Effects)
    V: List of variables of interest
    T: List of times of interest
    S: List of variables with equal time stamp
1. For each $r \in R$ find all the paths leading to a $f \in F$.
   Apply this step for all $f \in F$.
2. Add a unique subscript to each node in an individual
   path obtained in Step 1.
3. Represent each path as a PG where a node in the path
   becomes a vertex and a delay d ($d > 0$) on an arc
   between two vertices v1, v2 becomes
   Length(v1,v2)=d in the PG.
4. For each set of vertices in PG that represent a root
   node in TIN, add a temporal equality constraint
   'Equal' among its elements.
5. For each element v in V,
   (i) Find its subscript at the corresponding time t
       ($t \in T$) in G1. Let s be the subscript.
   (ii) Add variable v with subscript s to S.
6. Add temporal equality constraint 'Equals' among the
   elements of list S.
7. Construct an aggregate PG G2 using temporal
   statements provided in Steps 3-6.
```

The PG resulted from the temporal statements provided in Steps 2-6 is shown in Figure 3.5.10. The PG can now be used to answer the query the system modeler is interested in. For instance, the length between points representing events B and E is 5 time units. Therefore, in order to combine the impacts that affect node C at time 8 and 9, B must be executed 5 time units before E.

The what-if analysis not only identifies the temporal relationships that should exist between two actionable events in order to achieve a desired impact at a certain time instant, but it also tells a system modeler if the given set of requirements are temporally inconsistent.

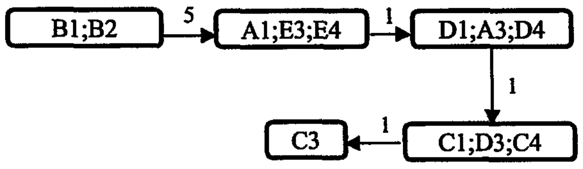


Figure 3.5.10. PG Used for What-If Analysis

3.6 PIL for Spatial Programming

This section presents some of the recent work done on PIL for spatial knowledge representation and reasoning. Hussain and Zaidi (2002) made the first attempt on applying the point-interval formalism for modeling spatial aspects of a set of objects defined in a one-dimensional world. Zaidi et al. (2003), and a recent work [Zaidi, 2004] extend the approach for a two-dimensional spatial system. This application of temporal constraint satisfaction problem to model spatial reasoning is similar to the way others in the spatial reasoning community [Rendal et al, in 1992; Renz and Nebel, 1998; Mortaz et al., 2000] have used Interval Algebra to model and reason about spatial information. Table 3.6.1 lists the PIL relations, function names, and their corresponding spatial equivalents. Using the new terminology, spatial objects, i.e., points and objects occupying space (interval,) can be modeled with the help of PIL statements.

The spatial version of PIL is referred to as Point Interval Spatial Logic, or PISL. In addition to the spatial relations in Table 3.6.1, PISL is added with orientation relations (Definition 3.6.1)

Definition 3.6.1: Set of Orientation Relations, O

O represents the set of orientation relations O*i* and is given as

O = {Pointing_right, Pointing_left}

The orientation relations are incorporated in the approach by appending a 'h' parameter to one of the start and end points of an interval. Table 3.6.2 lists the possible cases of orientation relations for both points and intervals, and their corresponding analytical representation. In the PG representation, the parameter is added to the corresponding vertex in the graph. The verification mechanism, presented earlier, is also extended to look for anomalies introduced due to erroneous orientation information, e.g., an inteval with two orientations.

The following are the new axioms that are added to the formalism for incorporating orientation relations among spatial objects.

D. Orientation Axioms [Hussain and Zaidi, 2002]

(D.1) Pointing_right (X) $\wedge$ (Y Left_of X) $\rightarrow$ (X Pointing_away Y)

(D.2) Pointing_left (X) $\wedge$ (Y Left_of X) $\rightarrow$ (X Pointing_towards Y)

(D.3) Pointing_left (X) $\wedge$ (X Left_of Y) $\rightarrow$ (X Pointing_away Y)

(D.4) Pointing_right (X) $\wedge$ (X Left_of Y) $\rightarrow$ (X Pointing_towards Y)

Other possible relations used to capture orientation of spatial objects, such as Pointing_opposite and Pointing_same can also be derived from the axioms and the two basic orientation relations.

Spatial Logic (PISL-2D)

The paper by Zaidi et al. [Zaidi, Rizvi, and Hussain., 2003] further extends the spatial logic for 2-dimensional (2D) case. In a 2D version of PISL, an object's spatial features are represented as its projections on x- and y-axes. The projections themselves take the form of intervals and/or points on their respective axes. The PISL relations among intervals and points on either axis are represented with the help of PG representation. The inference mechanism, in the 2D case, first looks at the two PG representations to establish the spatial relations between the objects' projections. A second set of axioms is then used to infer the spatial relation between the objects in space.

Lexicon

The lexicon of PISL-2D consists of the following primitive symbols:

Primitive Spatial Objects: Points, Vertical Lines (VL), Horizontal lines (HL), and Rectangles (made of VLs and HLs as edges) are the only spatial objects allowed. The objects are defined for a 2-dimensional world.

X-Projections: The projections of spatial objects on X-axis as points and/or intervals.

Y-Projections: The projections of spatial objects on Y-axis as points and/or intervals.

Spatial Relations: The set of relations Rs is given as:

Rs = {Top_of, Left_of, TopLeft_of, TopRight_of }

Figure 3.6.1 shows the decomposition of all primitive spatial objects to their corresponding projected point/interval representations. The X's in the figure represent the primitive spatial objects allowed in the methodology. The solid box in part (d) of the figure represents the spatial object "Rectangle", with its corresponding interval projections on the two axes.

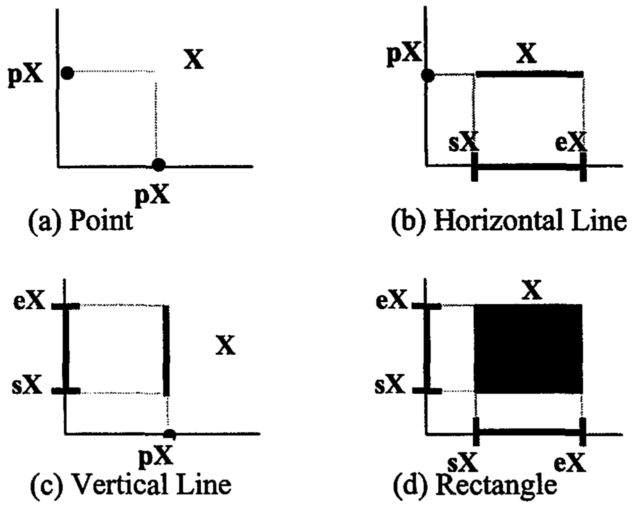


Figure 3.6.1. Primitive Objects and Their Projections on the Coordinate Axes

Syntactic and Semantic Structure

The syntactic and semantic structure of the statements in PISL is shown in Figures 3.6.2-5. The figures present all 16 combinations of the 4 primitive spatial objects for each of the spatial relation in Rs. The solid boxes shown in the figure represent the "Rectangle" spatial objects. A more accurate analytical representation for each of the spatial situation, shown in Figures 3.6.2-5, is given in Table 3.6.3. A number of other topological (i.e. RCC-8 [Randell et al., 1992],) and orientation relations can also be added to the set Rs and can be used to model the spatial specification.

Spatial Inference Engine (SPINE)

A spatial relation between two spatial objects can be described with the help of two sets of algebraic inequalities, shown in Table 3.6.3, with each set representing relationships among objects' projections on a coordinate axis. The inequalities in the table can be further refined to reflect the actual spatial situation. The inequalities on a particular axis represent the analytical model of PIL statements shown in Tables 3.1.3-5. A PG is constructed for each set of inequalities using the approach presented.

The inference mechanism of PISL, as implemented in SPINE, tries to establish the analytic representation between a pair of spatial objects by first performing searches in the two PGs. The resulting inequalities, if any, between objects' X- and Y-projection are matched with the ones in Table 3.6.3. The corresponding spatial relation, or relations for partial matches, is the inference made by the inference engine. A separate explicit composition table for this purpose can also be constructed that can help implement the axiomatic system of PISL-2D. A part of such a table is shown in Table 3.6.4. The legend for the abbreviated symbols used in the table is given at the bottom of the table.

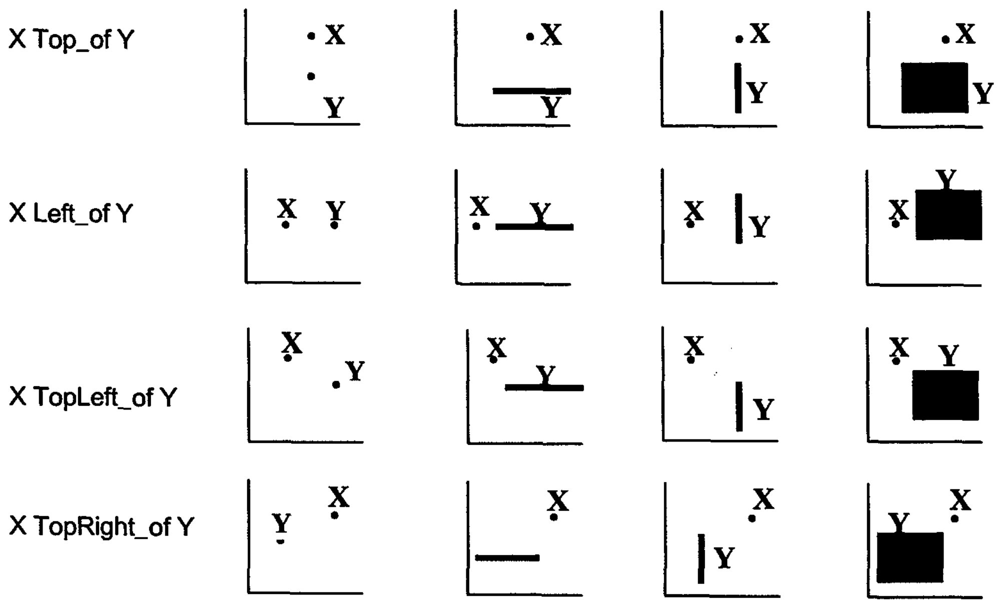


Figure 3.6.2. Spatial Relations Between a Point and other Spatial Objects

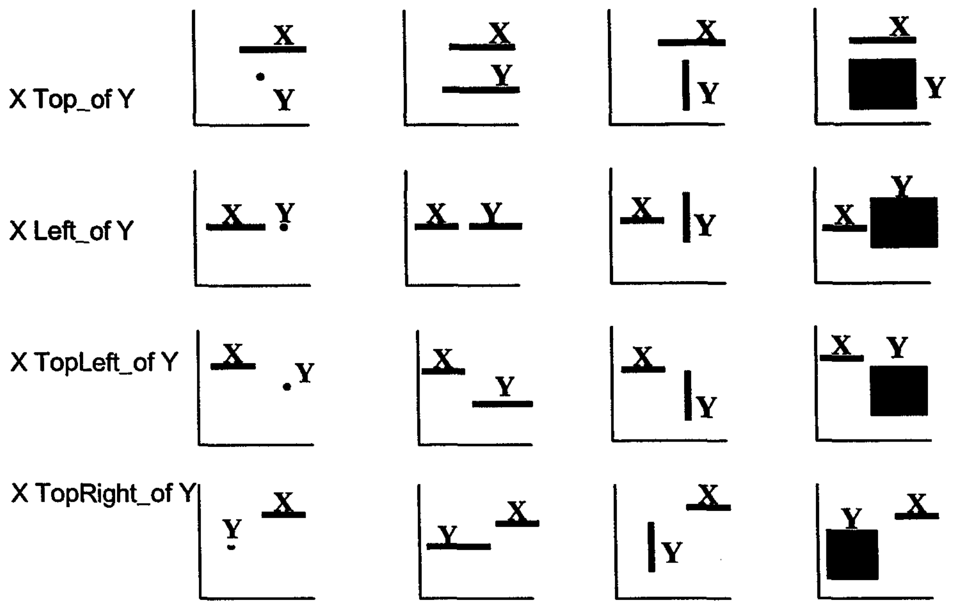


Figure 3.6.3. Spatial Relations Between a Horizontal Line and other Spatial Objects

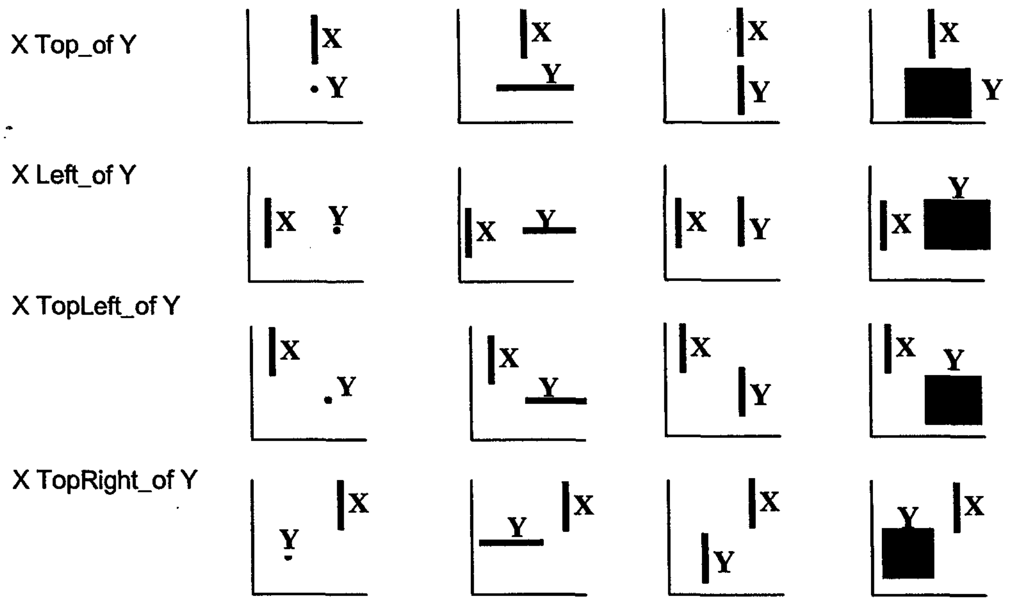


Figure 3.6.4. Spatial Relations Between a Vertical Line and other Spatial Objects

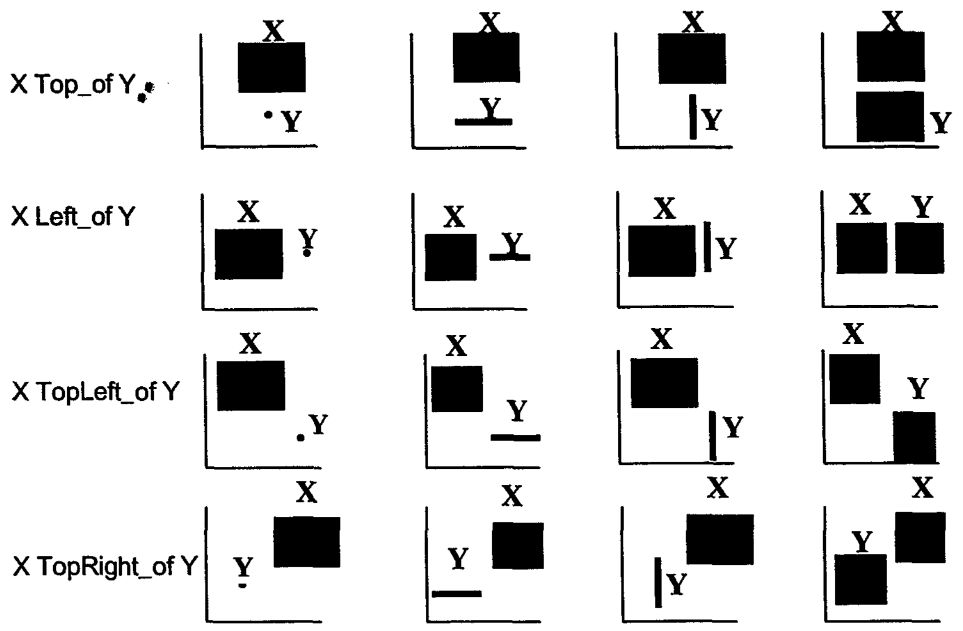


Figure 3.6.5. Spatial Relations Between a Rectangle and other Spatial Objects

<u>Example 3.6.1</u>

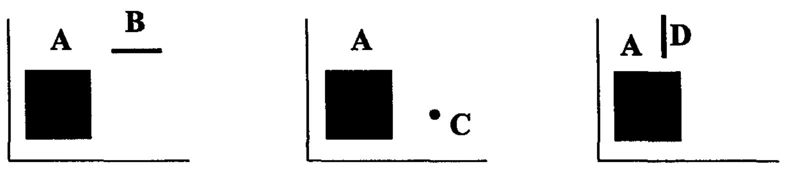
Figure 3.6.6 illustrates the methodology with the help of an example. Part (a) of the figure specifies four spatial objects, labeled as A, B, C, and D. The spatial relations between some of these objects are modeled with PISL statements. A pictorial representation of the spatial situation is depicted in part (b) of the figure. The PGs constructed for the objects' projections are shown in parts (c) and (d). The inference mechanism is illustrated in part (e), which establishes a spatial relation between two objects that is not explicitly given in the input.

(a) Statements in PISL

A Rectangle, B Horizontal Line, C Point, D Vertical Line

B TopRight_of A   A Left_of C   D Top_of A

(b) Spatial Situation Represented by the Statements



(c) PGx—Unified PG for X-Projections of the Objects

$$\begin{tikzcd} \boxed{sA} \arrow[rr] \arrow[dr] & & \boxed{eA} \arrow[r] \arrow[dr] & \boxed{sB} \arrow[r] & \boxed{eB} \\ & \boxed{pD} \arrow[ur] & & \boxed{pC} & \end{tikzcd}$$

(d) PGy—Unified PG for Y-Projections of the Objects

$$\begin{tikzcd} \boxed{sA} \arrow[rr] \arrow[dr] & & \boxed{eA} \arrow[r] \arrow[dr] & \boxed{sD} \arrow[r] & \boxed{eD} \\ & \boxed{pC} \arrow[ur] & & \boxed{pB} & \end{tikzcd}$$

(e) Query D ? C

Step 1: $pD < pC$ (in PGx)

Step 2: $pC < sD$ (in PGy)

Step 3: (Table Lookup) D TopLeft_of C

(f) Corresponding Axiom of PISL-2D:

If $(pD < pC)_{\text{on X-Axis}}$ AND $(pC < D)_{\text{on Y-Axis}}$ then $(D \text{ TopLeft\_of } C)_{\text{in Space}}$

Figure 3.6.6. Illustration of SPINE

Table 3.6.1. Spatial Interpretation of PIL Relations

PIL Relation/Function Name	**Corresponding Spatial Equivalent**
<	Left_of
M	Adjacent
O	Overlaps
S	Touches_left
D	Inside
F	Touches_right
≡	Same
Stamp	Coordinate
Length	Length/Distance

Table 3.6.2. Orientation Relations in PISL

Orientation Relation X inteval, Y point	**PISL Representation**
Pointing_right (X)	X = [sX, h:eX]
Pointing_left (X)	X = [h:sX, eX]
Pointing_right (Y)	X = [pX, h:pX]
Pointing_left (Y)	X = [h:pX, pX]

Table 3.6.3. Analytical Representation of PISL Relations

X = Point		**Y =**			
Spatial Relation	**Projection**	**Point**	**Horizontal Line**	**Vertical Line**	**Rectangle**
X Top_of Y	X-Axis	$pX = pY$	$sY \le pX \le eY$	$pX = pY$	$sY \le pX \le eY$
	Y-Axis	$pY \le pX$	$pY \le pX$	$sY \le pX$	$sY \le pX$
X Left_of Y	X-Axis	$pX < pY$	$pX < sY$	$pX < pY$	$pX < sY$
	Y-Axis	$pX = pY$	$pX = pY$	$sY \le pX \le eY$	$sY \le pX \le eY$
X TopLeft_of Y	X-Axis	$pX < pY$	$pX < sY$	$pX < pY$	$pX < sY$
	Y-Axis	$pY < pX$	$pY < pX$	$eY < pX$	$eY < pX$
X TopRight_of Y	X-Axis	$pY < pX$	$eY < pX$	$pY < pX$	$eY < pX$
	Y-Axis	$pY < pX$	$pY < pX$	$eY < pX$	$eY < pX$
X = Horizontal Line					
X Top_of Y	X-Axis	$sX \le pY \le eX$	$sY \le p \le eY$ $p \in \{sX, eX\}$	$sX \le pY \le eX$	$sY \le p \le eY$ $p \in \{sX, eX\}$
	Y-Axis	$pY \le pX$	$pY \le pX$	$sY \le pX$	$sY \le pX$
X Left_of Y	X-Axis	$eX < pY$	$eX < sY$	$eX < pY$	$eX < sY$
	Y-Axis	$pX = pY$	$pX = pY$	$sY \le pX \le eY$	$sY \le pX \le eY$
X TopLeft_of Y	X-Axis	$eX < pY$	$eX < sY$	$eX < pY$	$eX < sY$
	Y-Axis	$pY < pX$	$pY < pX$	$eY < pX$	$eY < pX$
X TopRight_of Y	X-Axis	$pY < sX$	$eY < sX$	$pY < sX$	$eY < sX$
	Y-Axis	$pY < pX$	$pY < pX$	$eY < pX$	$eY < pX$
X = Vertical Line					
X Top_of Y	X-Axis	$pX = pY$	$sY \le pX \le eY$	$pX = pY$	$sY \le pX \le eY$
	Y-Axis	$pY < eX$	$pY < eX$	$sY < eX$	$sY < eX$

X Left_of Y	X-Axis	$pX < pY$	$pX < sY$	$pX < pY$	$pX < sY$
	Y-Axis	$sX \leq pY \leq eX$	$sX \leq pY \leq eX$	$sY \leq p \leq eY$	$sY \leq p \leq eY$
				$p \in \{sX, eX\}$	$p \in \{sX, eX\}$
X TopLeft_of Y	X-Axis	$pX < pY$	$pX < sY$	$pX < pY$	$pX < sY$
	Y-Axis	$pY < sX$	$pY < sX$	$eY < sX$	$eY < sX$
X TopRight_of Y	X-Axis	$pY < pX$	$eY < pX$	$pY < pX$	$eY < pX$
	Y-Axis	$pY < sX$	$pY < sX$	$eY < sX$	$eY < sX$
X = Rectangle					
X Top_of Y	X-Axis	$sX \leq pY \leq eX$	$sY \leq p \leq eY$	$sX \leq pY \leq eX$	$sY \leq p \leq eY$
			$p \in \{sX, eX\}$		$p \in \{sX, eX\}$
	Y-Axis	$pY < eX$	$pY < eX$	$sY < eX$	$sY < eX$
X Left_of Y	X-Axis	$eX < pY$	$eX < sY$	$eX < pY$	$eX < sY$
	Y-Axis	$sX \leq pY \leq eX$	$sX \leq pY \leq eX$	$sY \leq p \leq eY$	$sY \leq p \leq eY$
				$p \in \{sX, eX\}$	$p \in \{sX, eX\}$
X TopLeft_of Y	X-Axis	$eX < pY$	$eX < sY$	$eX < pY$	$eX < sY$
	Y-Axis	$pY < sX$	$pY < sX$	$eY < sX$	$eY < sX$
X TopRight_of Y	X-Axis	$pY < sX$	$eY < sX$	$pY < sX$	$eY < sX$
	Y-Axis	$pY < sX$	$pY < sX$	$eY < sX$	$eY < sX$

Table 3.6.4. A Part of the Composition Table for PISL-2D Relations

	X Axis	Point (X) Point (Y)			
Y Axis		$X \equiv Y$	$X < Y$	$X <^{-1} Y$	$X ? Y$
Point (X) Point (Y)	$X \equiv Y$	T	L	L^{-1}	LTL^{-1}
	$X < Y$	T^{-1}	Tl^{-1}	Tr^{-1}	$Tr^{-1}T^{-1}Tl^{-1}$
	$X <^{-1} Y$	T	Tl	Tr	$TrTTl$
	$X ? Y$	TT^{-1}	$LTlTl^{-1}$	$TrTr^{-1}L^{-1}$	?
Point (X) Interval (Y)	$X < Y$	T^{-1}	Tr^{-1}	Tl^{-1}	$T^{-1}Tr^{-1}Tl^{-1}$
	$X s Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X d Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X f Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X <^{-1} Y$	T	Tl	Tr	$TTlTr$
	$X ? Y$	TT^{-1}	$Tr^{-1}LTl$	$Tl^{-1}L^{-1}Tr$	?
Interval (X) Interval (Y)	$X < Y$	T^{-1}	Tr^{-1}	Tl^{-1}	$T^{-1}Tl^{-1}Tr^{-1}$
	$X m Y$	T^{-1}	L	L^{-1}	$T^{-1}LL^{-1}$
	$X o Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X s Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X d Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X f Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X \equiv Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X <^{-1} Y$	T	Tl	Tr	$TTlTr$
	$X m^{-1} Y$	T	L	L^{-1}	TLL^{-1}
	$X o^{-1} Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X s^{-1} Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X d^{-1} Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X f^{-1} Y$	TT^{-1}	L	L^{-1}	$TT^{-1}LL^{-1}$
	$X ? Y$	TT^{-1}	$TlLTr^{-1}$	$Tl^{-1}L^{-1}Tr$	?

Legend

- ($X Ri^{-1} Y$) is equivalent to ($Y Ri X$)
- T = Top_of
- L = Left_of
- Tl = TopLeft_of
- Tr = TopRight_of
- A string of spatial relations represents a disjunction among the atomic relations, e.g., ($X TT^{-1} Y$) is equivalent to ($X T Y$) $\vee$ ($Y T X$)

3.7 The Software Implementation

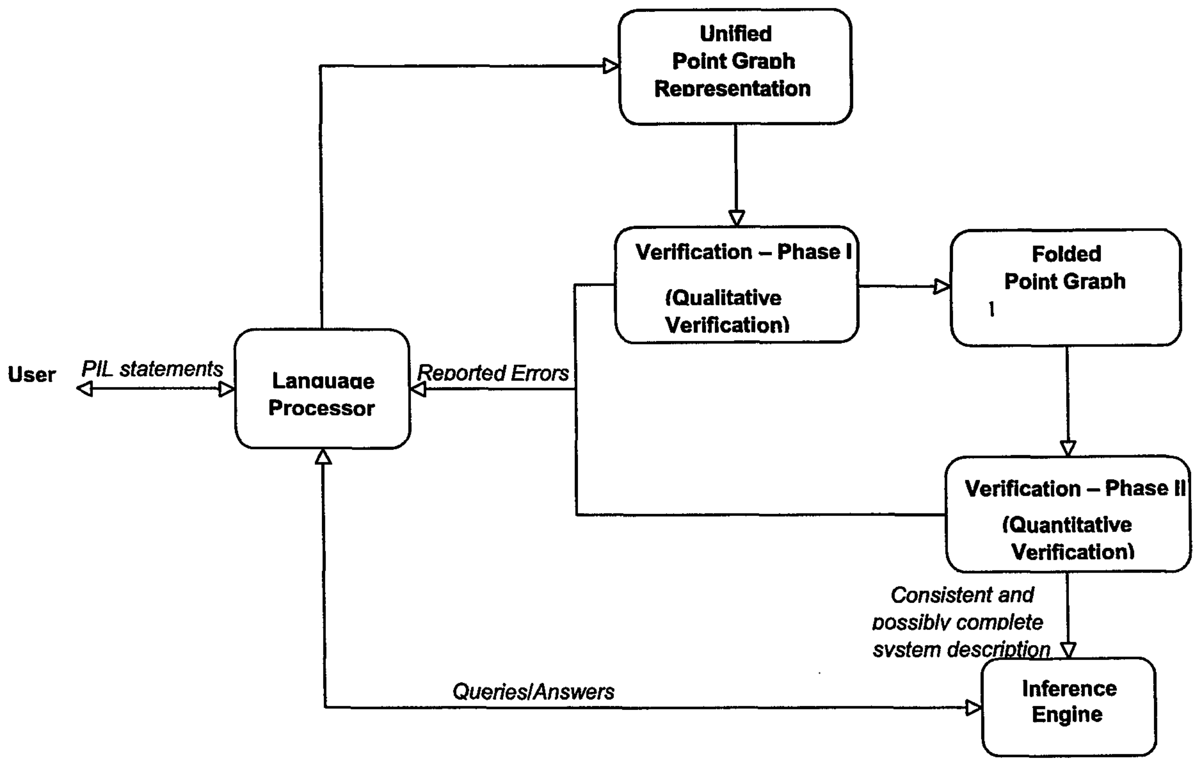

Unified Point Graph Representation
Verification – Phase I (Qualitative Verification)
Folded Point Graph
User
PIL statements
Language Processor
Reported Errors
Verification – Phase II (Quantitative Verification)
Consistent and possibly complete system description
Queries/Answers
Inference Engine

Figure 3.7.1. The software Architecture

The ML Implementation

The ML implementation of Point-Interval Logic and its inference engine was discontinued in the year 2003 after its native platform, DesignCPN, was upgraded to CPNTools. The upgrade CPNTools does not support the features of ML language used in the PIL implementation. A decision, therefore, was taken to migrate to Windows based implementation of the software.

The .NET Version

The recent implementation of Point-Interval Logic has been in the form of a .NET class library (an API – application programming interface) called PIL Engine (pilengine.dll). It can be used in any programming language for .NET platform. PILClient.exe is the user interface of PIL Engine API. It can also be used as a reference on implementing customized front-end for PIL Engine.

Both of these pieces of software are downloadable from http://viking.gmu.edu/PILEngine.htm along with the documentation on how to use PIL Engine.

PIL Engine

PIL Engine is the Point-Interval Logic API. It accepts as input PIL statements, checks these statements for consistency and then constructs a point graph representation. Once point graph is constructed the PIL Engine can be used to infer new relationships between different PIL entities. The following is the sequence of steps required for using PIL Engine in an application.

1. Add a reference of PIL Engine (pilengine.dll) to the client application using Solution Explorer of the .NET IDE.
2. Add PIL namespace to the application namespaces.

   using PIL;
3. Create an instance of PIL Engine.

   PILEngine pilEngine = new PILEngine ();
4. Declare all variables to be used in PIL statements.

   pilEngine.addPoint (string strVariableName);

   pilEngine.addInterval (string strVariableName);
5. Add PIL statements.

   pilEngine.addStampStatement (string strPoint, double dStamp);

   pilEngine.addLengthStatement (string strStart, string strEnd, double dLength);

   pilEngine.addStatement (string strVariable1, string strVariable2, string strRelation);
6. Construct the point graph using this construct method of PIL Engine. In case there is no inconsistency in the PIL statements, the method will return true.

   pilEngine.construct ();
7. Once the point graph has been constructed it can be revised to accommodate any changes in the PIL statements. During revision new PIL statements can be added, previously added PIL statements can be deleted or modified (modify means delete followed by add).

   pilEngine.deleteStampStatement (string strPoint);

   pilEngine.deleteLengthStatement (string strStart, string strEnd);

   pilEngine.deleteStatement (string strVariable1, string strVariable2);

8. After revision the point graph must be constructed again before any inferences can be made.
9. Once the point graph has been constructed without errors it can be queried for various inferences. There are four kinds of queries at present: query stamp, query length, query relation, query if relation exists.

   dStamp = pilEngine.queryStamp (string strPoint)

   dLength = pilEngine.queryLength (string strStart, string strEnd)

   strRelation = pilEngine.queryRelation (string strVariable1, string strVariable2);

   bRelationExists = pilEngine.queryRelation (string strVariable1, string strVariable2, string strRelation);

PIL Client

PILClient.exe is the user interface of PIL Engine API.

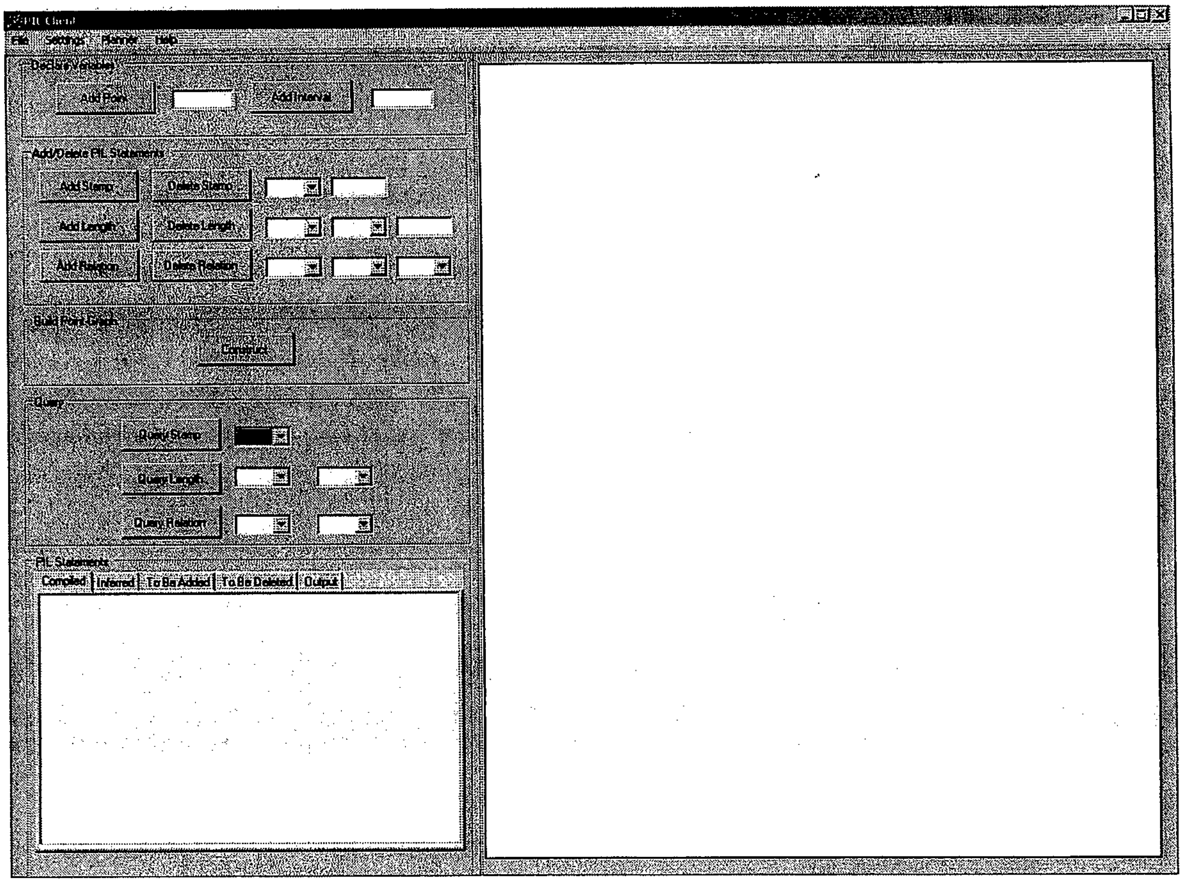
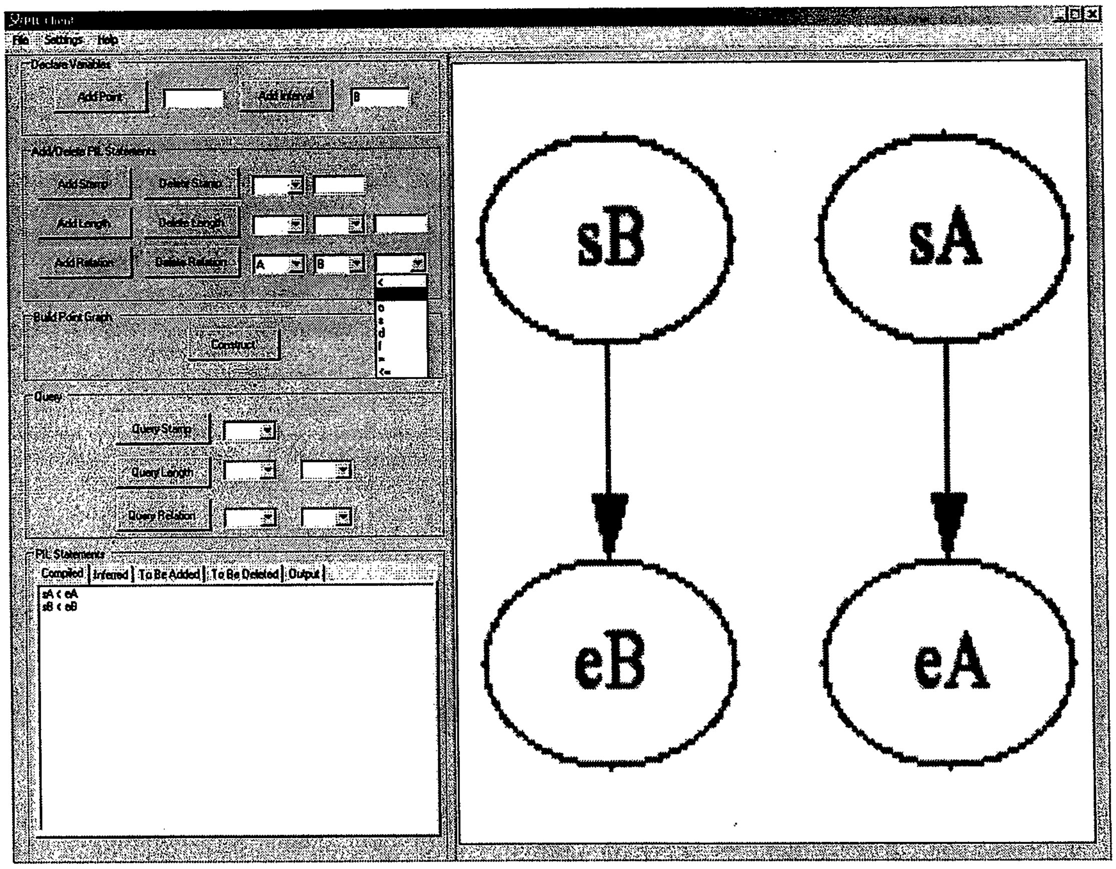


Figure 3.7.2. The PIL Client User Interface

The following sequence of steps is required for entering PIL statement into PIL Client.

1. Input Points and Intervals: Type the name of the points and intervals involved in the PIL statements in the text boxes next to *Add Point* and *Add Interval* buttons respectively and then click the button to add. Note when an interval is added (say X); the points marking start (sX) and the end (eX) of the interval and the relation sX < eX are automatically added.
2. Input PIL Statements: To assign a stamp to a point, select the point from the drop-down list, enter the stamp and press the *Add Stamp* button. To assign length to an interval, select the starting and the ending points of the interval from the drop-down lists, enter length and then press the button *Add Length*. To add a PIL relation between two PIL entities, select the two entities, select the relation to be entered and then press *Add Relation*.
3. Construct the Point Graph: After adding the PIL statements, construct the point graph by pressing the *Construct* button. If the entered PIL statements contain inconsistencies, a message will appear displaying the error; otherwise, the PIL statements are consistent. The constructed point graph will also be



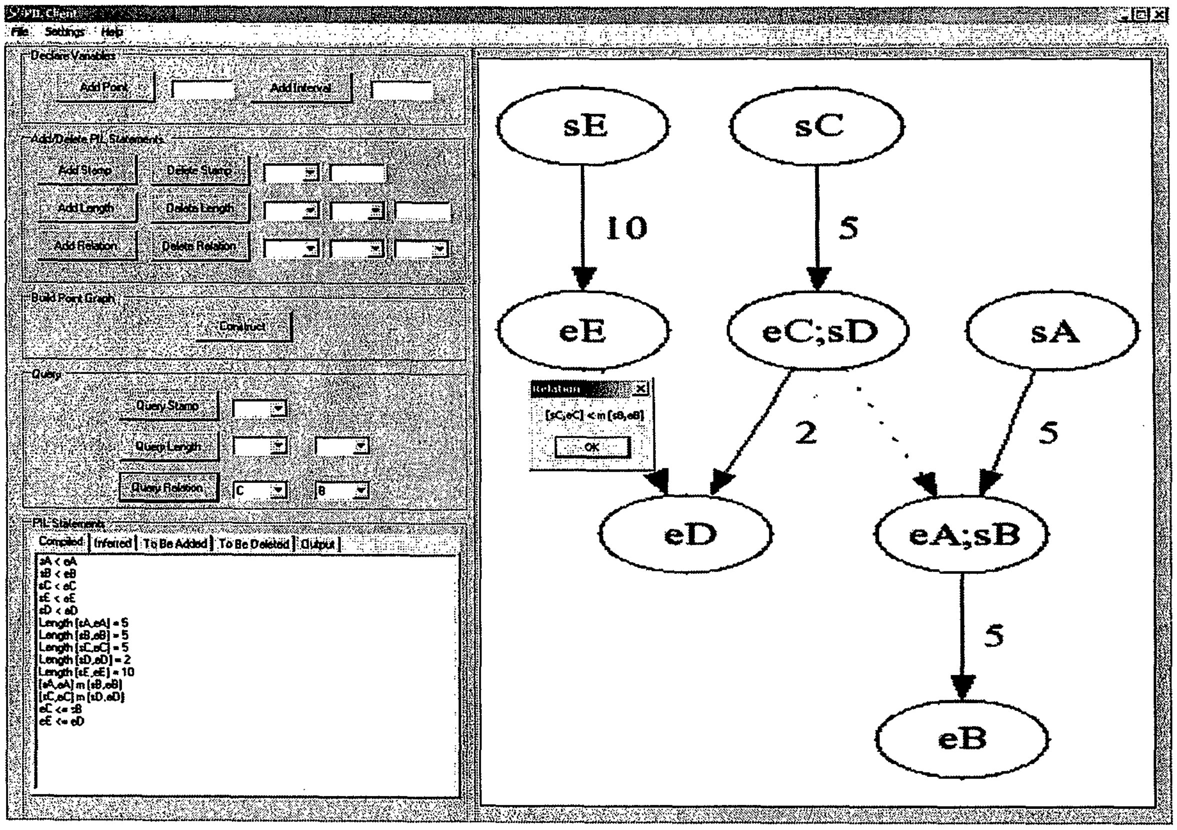
```
sA < eA
sB < eB
```

displayed.

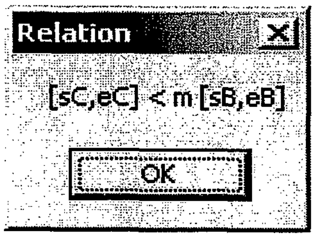
Figure 3.7.3. The Constructed Point Graph

4. Query the Point Graph: To query the stamp of a point, select the point from the list and press *Query Stamp*. The stamp of the point will be displayed. A "–1" shows that the time stamp of the point cannot be inferred using input PIL statements. To query the length of an interval, select its start and end points from the list and press *Query Length*. The length of the interval will be displayed. A "–1" shows that the length of the interval cannot be inferred using input PIL statements. To query the relation between two PIL entities, select the two entities and press *Query Relation*. The relation between the two entities will be displayed. A "?" represents unknown relationship. When "i" is suffixed to a relation it represents inverse relation. For example "A fi B" means "B f A".
5. Revise the Point Graph: Once the point graph has been constructed it can be revised to accommodate any changes in the PIL statements. During revision new PIL statements

can be added, previously added PIL statements can be deleted or modified (modify means delete followed by add). To delete the stamp assigned to a point, select the point from the drop-down list and press the *Delete Stamp* button. To delete the length assigned to an interval, select the starting and the ending points of the interval from the drop-down lists and press the button *Delete Length.* To delete the PIL relation between two PIL entities, select the two entities and press *Delete Relation* button. To modify a PIL statement: first delete the old statement and then add the new modified statement.



```
sA < eA
sB < eB
sC < eC
sE < eE
sD < eD
Length [sA,eA] = 5
Length [sB,eB] = 5
Length [sC,eC] = 5
Length [sD,eD] = 2
Length [sE,eE] = 10
[sA,eA] m [sB,eB]
[sC,eC] m [sD,eD]
eC <= sB
eE <= eD
```


Relation

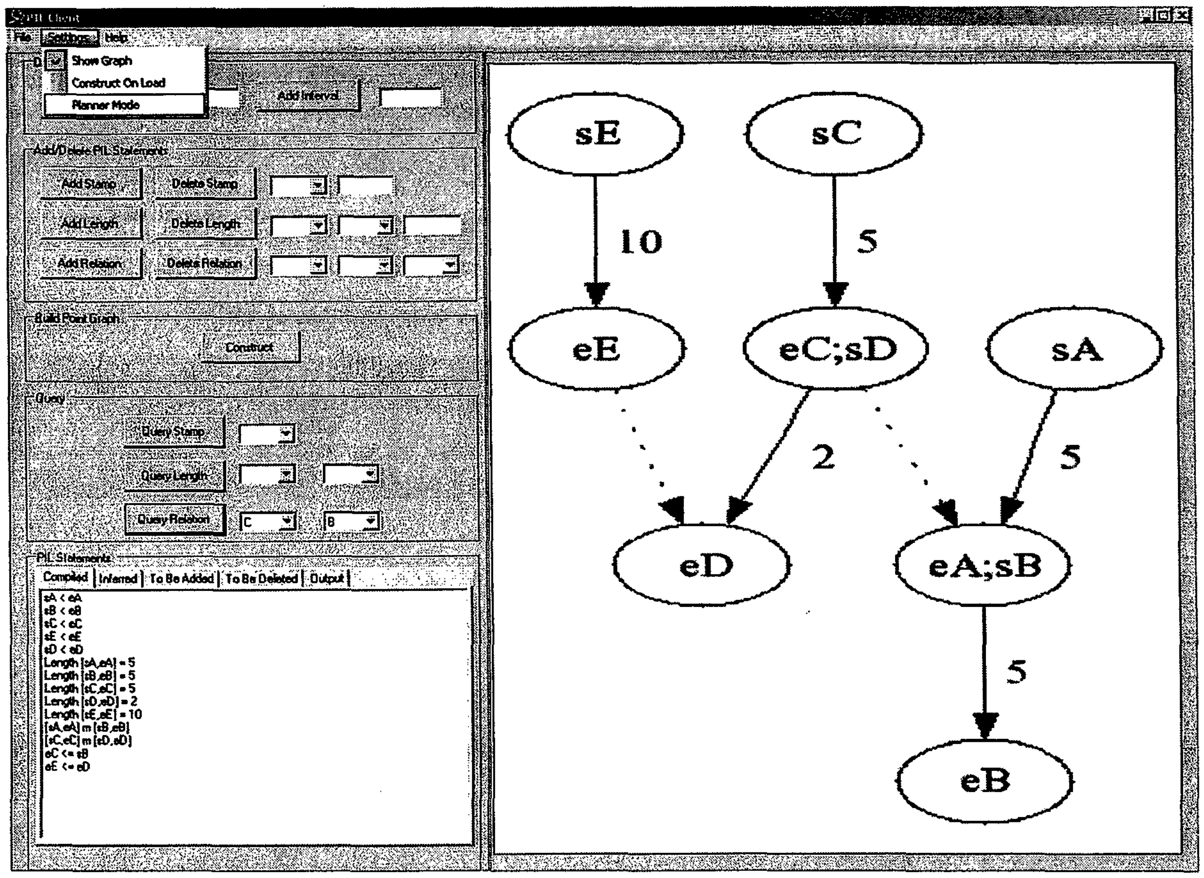
[sC,eC] < m [sB,eB]

OK

Figure 3.7.4. Invoking the Inference Engine

Planner Mode

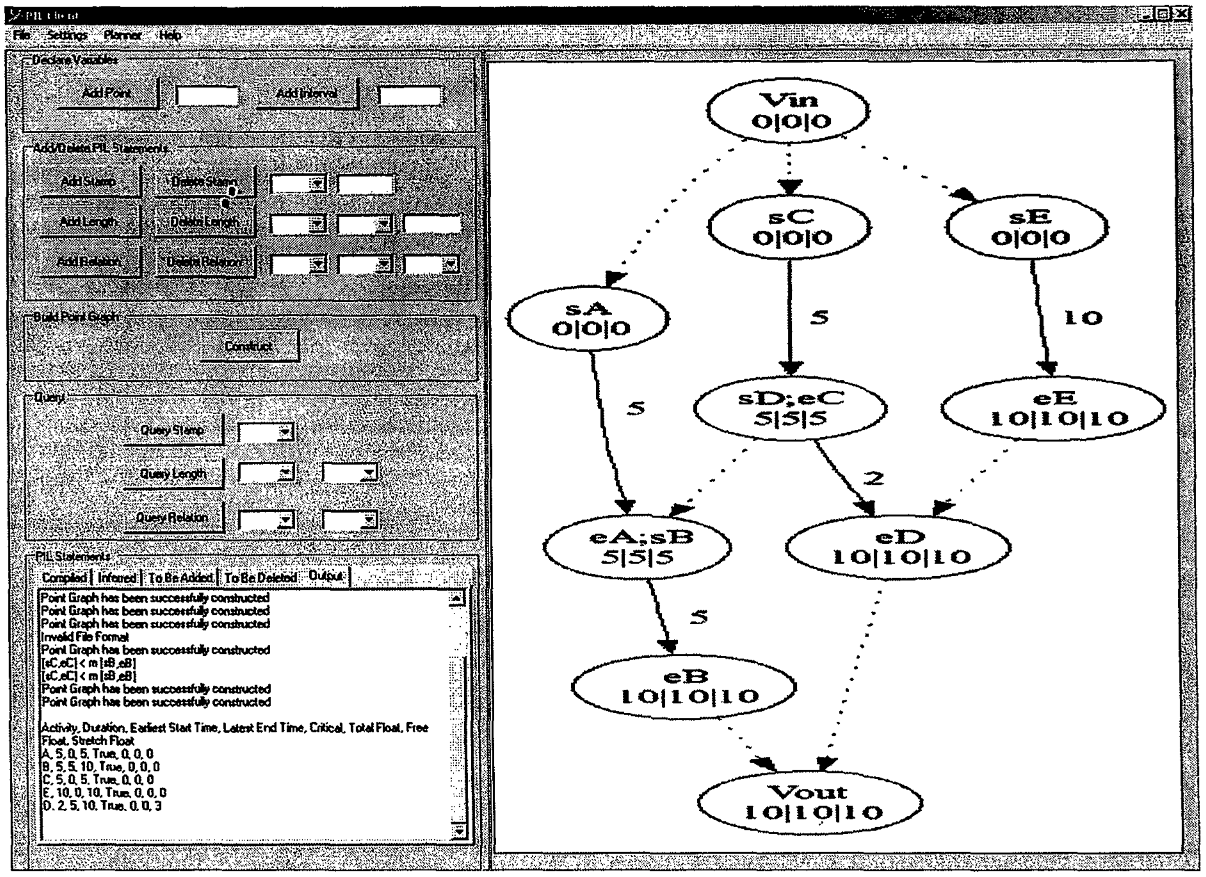
To switch to planner mode select it from the *Settings* menu.



```
sA < eA
sB < eB
sC < eC
sE < eE
sD < eD
Length [sA,eA] = 5
Length [sB,eB] = 5
Length [sC,eC] = 5
Length [sD,eD] = 2
Length [sE,eE] = 10
[sA,eA] m [sB,eB]
[sC,eC] m [sD,eD]
eC <= sB
eE <= eD
```

Figure 3.7.5. The Planner Mode

Switching the mode closes the currently opened file. Open the PIL file containing plan's specifications. Then use *Construct* button to generate the point graph.

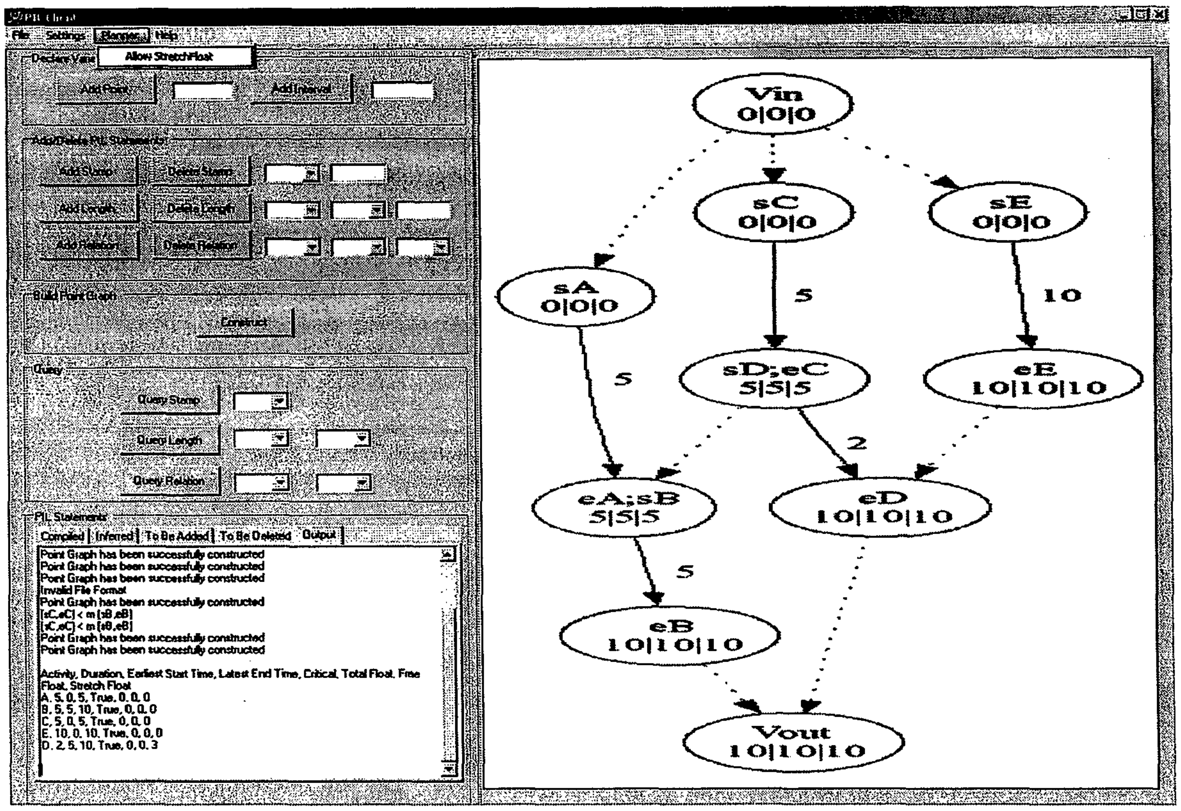


```
Point Graph has been successfully constructed
Point Graph has been successfully constructed
Point Graph has been successfully constructed
Invalid File Format
Point Graph has been successfully constructed
[sC,eC] < m [sB,eB]
[sC,eC] < m [sB,eB]
Point Graph has been successfully constructed
Point Graph has been successfully constructed

Activity, Duration, Earliest Start Time, Latest End Time, Critical, Total Float, Free
Float, Stretch Float
A, 5, 0, 5, True, 0, 0, 0
B, 5, 5, 10, True, 0, 0, 0
C, 5, 0, 5, True, 0, 0, 0
E, 10, 0, 10, True, 0, 0, 0
D, 2, 5, 10, True, 0, 0, 3
```

Figure 3.7.6. The Constructed Point Graph (Planner Mode)

Each node on the graph displays Ev|Lv|Tv (Earliest Start Time | Late Start Time | Latest Start Time) and Tv of Vout represents the completion time of the mission. The output window displays a table describing various floats/slacks. If Stretch Float is not to be allowed it can be turned off by clicking *Allow StretchFloat* from the *Planner* menu.

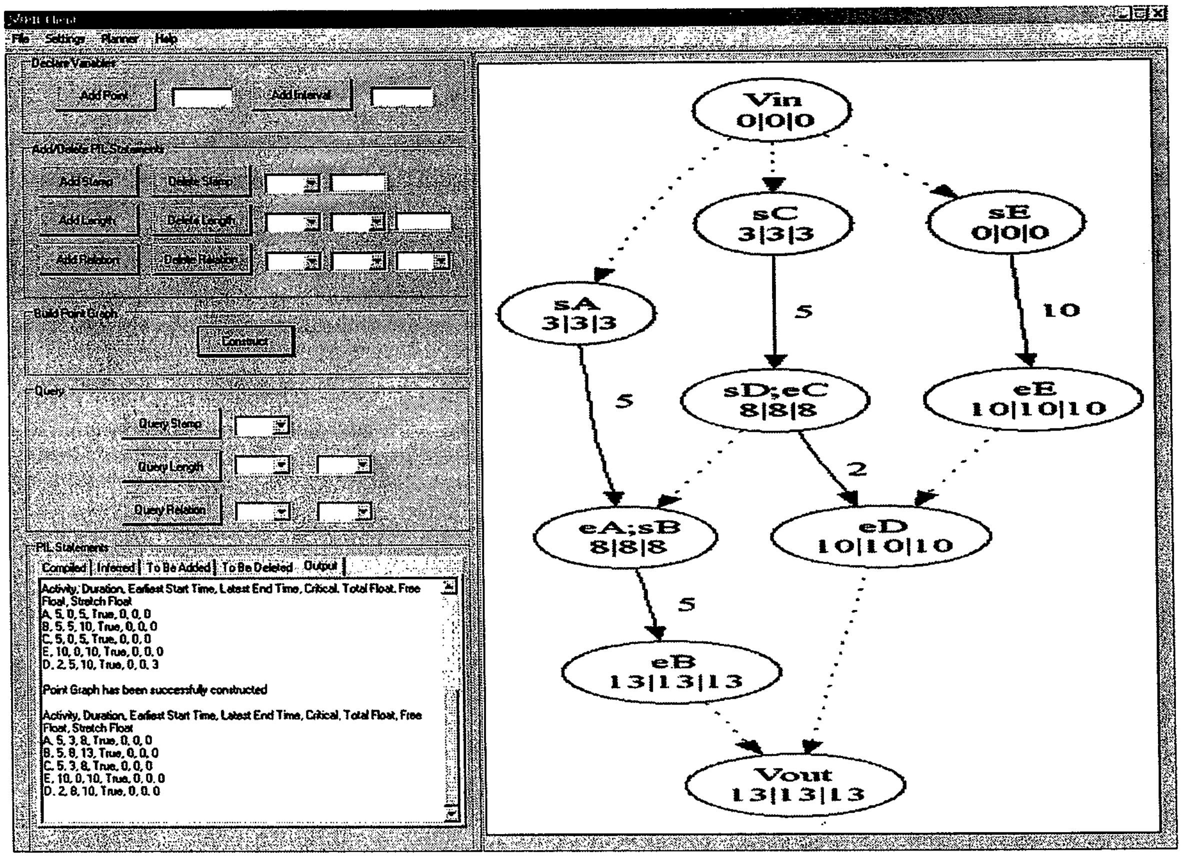


```
Point Graph has been successfully constructed
Point Graph has been successfully constructed
Point Graph has been successfully constructed
Invalid File Format
Point Graph has been successfully constructed
[sC,eC] < m [sB,eB]
[sC,eC] < m [sB,eB]
Point Graph has been successfully constructed
Point Graph has been successfully constructed

Activity, Duration, Earliest Start Time, Latest End Time, Critical, Total Float, Free
Float, Stretch Float
A, 5, 0, 5, True, 0, 0, 0
B, 5, 5, 10, True, 0, 0, 0
C, 5, 0, 5, True, 0, 0, 0
E, 10, 0, 10, True, 0, 0, 0
D, 2, 5, 10, True, 0, 0, 3
```

Figure 3.7.7. Disallowing StretchFloat

Again use the *Construct* button to recalculate Ev|Lv|Tv.



```
Activity, Duration, Earliest Start Time, Latest End Time, Critical, Total Float, Free
Float, Stretch Float
A, 5, 0, 5, True, 0, 0, 0
B, 5, 5, 10, True, 0, 0, 0
C, 5, 0, 5, True, 0, 0, 0
E, 10, 0, 10, True, 0, 0, 0
D, 2, 5, 10, True, 0, 0, 3

Point Graph has been successfully constructed

Activity, Duration, Earliest Start Time, Latest End Time, Critical, Total Float, Free
Float, Stretch Float
A, 5, 3, 8, True, 0, 0, 0
B, 5, 8, 13, True, 0, 0, 0
C, 5, 3, 8, True, 0, 0, 0
E, 10, 0, 10, True, 0, 0, 0
D, 2, 8, 10, True, 0, 0, 0
```

Figure 3.7.8. The Constructed Point Graph (without StretchFloat)

3.8 Conclusion

The PIL formalism has been shown to incorporate temporal and spatial information separately from each other. The present implementation offers a toolkit of graph-based algorithms for implementing inference and verification mechanisms, revision algorithms for identifying the type of change before making the revision to the plan, and the application of the logic for temporal and spatial knowledge representation and reasoning. A recent attempt on combining the graph-based revision and the inference mechanisms into an algorithm that can efficiently handle the dynamic change has yielded promising results. The objectives of the future research are (a) to exploit the existing capabilities of graphical representation and efficient revision/inference algorithms to devise fast interactive spatio-temporal planners; and (b) to incorporate resource capabilities attributes within the new formalism to effectively address the interactive capability planning problem. The goal will be to develop a formalism that provides for interactive capability planning and to implement that formalism in a tool suite that can be used to support the development, execution, and assessment of robust plans and support the adaptation of those plans in a dynamic environment. While substantial progress has been achieved, there remain significant theoretical challenges to be addressed. Furthermore, changes that have occurred within the US Air Force and the DOD during the last few years have necessitated the reformulation of the problem of Air and Space Operations to the on-the-fly assemblage of capabilities to support operations in a joint and coalition environment.

The capability-driven interactive planning can be described as a constraint satisfaction problem, where the constraints capture the temporal, spatial, and resource attributes' requirements for the mission under consideration. The time-sensitive aspect requires a planner to sequence time intervals (or points) associated with mission activities, or services required, without violating any of the system specifications (or temporal constraints), given a priori. The spatial aspect, on the other hand, needs to look at the (spatial) availability of the physical resources capable of handling the required task list. The constraints on attributes of the available resources may result in alternates to the deployment of resources, where each alternate represents its own capability to handle the tasks. In most real world situations, the dynamic nature of the domain may require revising a produced model (or plan) during and/or after the system specification phase, e.g., the constraints or system/mission requirements or system capabilities may change during or before a plan's execution. The future challenge is to devise a formalism that has these additional properties:

(a) It can handle the three different types of information (temporal, spatial, and capability attributes associated with resources and their dynamics) within a single analytical formulation.

(b) It can develop robust plans with the help of analytic techniques that not only perform well under a specified set of scenarios, but can be changed (real-time) to accommodate new conditions with minimal perturbation.

(c) It should map to a graphical representation of system specifications capable of modeling (and reasoning about) information at different levels of abstraction.

(d) It should be able to separate domain knowledge from problem-solving knowledge. This requirement eliminates the need for a planner to learn the theoretical background of the formalism before using it for planning missions.

A human planner, using such formalism, should be able to interact with it by modifying system/mission specifications and be able to generate alternate plans with different capability measures for each.

References

Abadi, M., and Manna, Z. (1985). Nonclausal temporal deduction, *Logics of Programs, Lecture Notes in Computer Science*, R. Parikh, Ed. Berlin, Germany: Springer-Verlag.

Allen, J. (1991). Formal Models of Planning. In *Readings in Planning*. Morgan Kaufmann, 50–54.

Allen, J. (1991). Planning as Temporal Reasoning. In *Proceedings of KR '91*, 3–14.

Allen, J. (1991). Temporal reasoning and planning. In J. Allen, H. Kautz, R. Pelavin & J. Tenenberg (eds.), *Reasoning About Plans*. San Mateo, CA: Morgan Kaufmann, 1–68.

Allen, J. and Ferguson, G. (1994). Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation* 4: 531–579.

Allen, J. and Koomen, J. (1983). Planning Using a Temporal World Model. In *Proceedings IJCAI '83*, 2.

Allen, J. and Hayes, P. (1985). A common-Sense Theory of Time. In *Proceedings IJCAI-85*, 1, 528–531.

Allen, J. F. (1983). Maintaining Knowledge About Temporal Intervals, *Communications of ACM,* 26, (1983), 832-843.

Allen, J. F. (1984). Towards a General Theory of Action and Time, *Artificial Intelligence,* 23(2), 123-154.

Allen, J.; Kautz, H.; Pelavin, R., and Tenenberg, J. (1991). *Reasoning About Plans*. SanMateo, CA: Morgan Kaufmann.

Anger, F. D.; Rodriguez, R. V.; Guesgen, H. W., and Van Benthem, J. (1996). Space, Time and Computation Trends and Problems, Appl. Intell., 6(1), pp 5-9.

Artale, A and Franconi, E. (2000). A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence,* 30: 171-210. Kluwer Academic Publishers.

Augusto, J. C. (2001). The Logical Approach to Temporal Reasoning. *Artificial Intelligence Review,* 16: 301-333. Kluwer Academic Publishers.

Bacchus, and Kabanza, F. (1996). Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA 1215-1222.

Barringer, H.; Fisher, M.; Gabbay, D., and Gough, G. (eds.) (2000). *Advances in Temporal Logic*, Applied logic series, Volume 16. Dordrecht, Boston: Kluwer Academic Publishers.

Bochman, A. (1990a). Concerted Instant-Interval Temporal Semantics I: Temporal Ontologies. Notre Dame Journal of Formal Logic 31(3): 403–414.

Bochman, A. (1990b). Concerted Instant-Interval Temporal Semantics II: Temporal Valuations and Logics of Change. *Notre Dame Journal of Formal Logic* 31(4): 581–601.

Busacker, R. G., and Saaty, T. L. (1965). *Finite Graphs and Networks: an introduction with applications*. New York, McGraw-Hill.

Cerro, Fari˜ as del (1985). Resolution modal logic, in *Logics and Models of Concurrent Programs*, K. R. Apt, Ed. Berlin, Germany: Springer-Verlag, pp. 27–56.

Cervesato, I; Franceschet, M., and Montanari, A. (1997a). A Hierarchy of Modal Event Calculi:Expressiveness and Complexity. In *Proceedings of the Second International Conference on Temporal Logic, ICTL'97*, 1–17.

Cervesato, I; Franceschet, M., and Montanari, A. (1997b). The Complexity of Model Checking in Modal Event Calculi. In *Proceedings of the Fourteenth International Conference on Logic Programming – ICLP'97.*

Cervesato, I; Franceschet, M., and Montanari, A. (1998). The Complexity of Model Checking in Modal Event Calculi with Quantifiers. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning – KR'98.*

Chellas, B.F. (1980). *Modal Logic. An Introduction.* Cambridge University Press.

Chittaro, L.; Goodwin, S.; Hamilton, H., and Montanari, A. (1996) (eds.) *Proceedings of the Third International Workshop on Temporal Representation and Reasoning.* Los Alamitos, California, USA: IEEE Computer Society Press.

Clarke, E. M. et al. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Trans. Progr. Lang. Syst.*, vol. 8, pp. 244–263.

Dechter, R.; Meiri, I., and Pearl, J. (1991). Temporal Constraint Network, *Artificial Intelligence,* 49, (1991), 61-95.

Davis E. (1986). Representing and Acquiring Geographic Knowledge, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986

Davis E. (1990). Representations of Commonsense Knowledge, Morgan Kaufmann Publishers, San Mateo, CA, 1990

Drakengren, T., and Jonsson, P. (1996). Maximal tractable subclassse of Allen's interval algebra: Preliminary report. In *Proc. AAAI'96*, 389-394.

Drakengren, T., and Jonsson, P. (1997a). Eight Maximal Tractable Subclasses of Allen's Algebra with Metric Time, *Journal of Artificial Intelligence Research,* 7, 25-45.

Drakengren T., and Jonsson, P. (1997b). Twenty-One Large Tractable Subclasses of Allen's Algebra, Artificial Intelligence, 93:297-319.

Drakengren, T., and Jonsson, P. (1997c). Towards a complete classification of tractability in Allen's algebra. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97)*, Nagoya, Japan.

Findlay, J. N. (1941). Time: A treatment of some puzzles, Aust. J. Phil., vol.19, pp. 216–235.

Gabbay, D., and Barringer, H. (eds.) (1997). *Proceedings of the Second International Conference on Temporal Logic.* Manchester, UK: Applied Logic Series. Kluwer.

Gabbay, D., and Ohlbach, H. (eds.) (1994). *Proceedings of the First International Conference on Temporal Logic.* Bonn, Germany: Springer Verlag.

Gabbay, D.; Hodkinson, I., and Reynolds, M. (1994). *Temporal Logic (Mathematical Foundations and Computational Aspects).* Oxford: Clarendon Press.

Galton, A (1987). Temporal Logics and Their Applications. New York: Academic Press.

Galton, A. (1990). A Critical Examination of Allen's Theory of Action and Time. *Artificial Intelligence* 42: 159–188.

Gerevini, A., and Schubert, L. (1993a). Efficient Temporal Reasoning through Timegraphs, in *Proceedings of IJCAI-93*.

Gerevini, A., and Schubert, L. (1993b). Temporal Reasoning in TimeGraph I-II, *SIGART Bulletin* 4(3).

Gerevini, A., and Schubert, L.K. (1995). Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence, 74(2):207--248.*

Ghallab, M., and Alaoui, A. M. (1989). Managing Efficiently Temporal Relations Through Indexed Spanning Trees. *IJCAI 1989: 1297-1303.*

Golumbic, M. C., and Shamir, R. (1992). Algorithms and Complexity for Reasoning about Time. In *Proceedings of the 10th National Conference on Artificial Intelligence*. AAAI, AAAI, Press/MIT Press, 741–747.

Golumbic, M. C., and Shamir, R. (1993). Complexity and Algorithms for Reasoning about Time: A Graph Theoretic Approach. *Journal of ACM*, 40(5), pp 1108-1133.

Goodwin, S., and Trudel, A. (eds.) (2000). *Proceedings of the Seventh International Workshop on Temporal Representation and Reasoning*. Cape Breton, Canada: IEEE Computer Society Press.

Guesgen, H. W.; Anger, F. D., and Ligozat, G. (2003). In *proceedings of the Workshop on Spatial and Temporal Reasoning,* Eighteenth IJCAI.

Guesgen, H. W.; Anger, F. D.; Ligozat, G., and Rogriquez, R. (2003). Special Issue on Spatial and Temporal Reasoning, *Journal of Universal Computer Science*, vol 9, Issue 9.

Hamblin, C. L. (1972). Instants and Intervals. In F. Haber J. Fraser & G. Muller (eds.), *The Study of Time*. New York: Springer Verlag, 324–328.

Jonsson, P., and Backstrom, C. (1996). A linear programming approach to temporal reasoning. In *Proceedings of AAAI-96*, 1235--241.

Kahn, K., and Gorry, G. (1977). Mechanizing temporal logic, *Artif. Intell.*, vol.9, pp. 87–108.

Kautz, H., and Ladkin, P. B. (1991). Integrating metric and qualitative temporal reasoning. In *Proceedings of AAAI-91*, Anaheim, CA, 241-246.

Kautz, H. (1999). Temporal Reasoning, In *The MIT Encyclopedia of Cognitive Science*, MIT Press, Cambridge.

Keretho, and Loganantharaj (1991). Qualitative and Quantitative Time Interval Constraint Networks, pp 239-246 ACM 089791-382-5/91/0003/0239 .

Koubarakis, M. (1992). Dense time and temporal constraints with 6=. In *Proceedings of KR-92,* Cambridge, Massachusetts, 24-35.

Koubarakis, M. (1995). From local to global consistency in temporal constraint networks. In *Proceedings of Principles and Practice of Constraint Programming,* - CP95, Cassi, France, 53-69.

Koubarakis, M. (1996). Tractable disjunctions of linear constraints. CP'96. Proc. of the 2nd Int. Conf. on *Principles and Practice of Constraint Programming,* 297--301.

Krokhin, A.; Jeavons, P., and Jonsson, P. (2003). Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. *Journal of the ACM,* 50(5), pp 591-640.

Ladkin, P. B., and Maddux, R. D. (1988). Representation and reasoning with convex time intervals, Tech. Report KES.U.88.2, Kestrel Institute.

Ladkin, P. and Maddux, R. (1994). On binary constraint problems. *Journal of the Association for Computing Machinery,* 41(3):435-469.

Ligozat, G. (1991). On generalized interval calculi. In *Proceedings AAAI-91,* Anaheim, Calafornia, 234-240.

Ligozat, G. (1996). A new proof of tractability for ORD-Horn relations. In *Proceedings of AAAI-96,* 395-401.

Ma, C. (1999). On Planning Time Sensitive Operations, *MS Thesis,* SE, George Mason University, VA, 1999.

Ma, J., and Knight, B. (2001). Reified Temporal Logics: An Overview. *Artificial Intelligence Review,* 15, 189-217.

Manna, Z., and Pnueli, A. (1981). Verification of concurrent programs: The temporal framework, *The Correctness Problem in Computer Science,* R. S. Boyer and J. S. Moore, Eds. London, U.K.: Academic, pp. 215- 273.

Martinez, J. and Silva, M. (1982). A Simple and Fast Algorithm to obtain all invariants of a generalized Petri Net, *Informatik-Fachbrichte,* 52, Springer Verlag, 301-310.

McArthur, R. P. (1976). *Tense Logic.* Dordrecht, The Netherlands: Reidel.

McDermott, D. (1982). A Temporal Logic for Reasoning About Processes and Plans, *Cognitive Science,* vol. 6, pp. 101-155, 1982.

Meiri, I. (1991). Combining Qualitative and Quantitative Constraints in Temporal Reasoning, in *Proc. of AAAI-91,* pp. 260-267, Anaheim, CA.

Meiri, I. (1996). Combining Qualitative and Quantitative Constraints in Temporal Reasoning, *Artificial Intelligence,* 87, 343-385.

Morris, R., and Khatib, L. (eds.) (1994). *Proceedings of the First International Workshop on Temporal Representation and Reasoning.* Daytona Beach, FL: Florida Artificial Intelligence Research Society.

Morris, R., and Khatib, L. (eds.) (1995). *Proceedings of the Second International Workshop on Temporal Representation and Reasoning.* Melbourne, FL: Florida Artificial Intelligence Research Society.

Morris, R., and Khatib, L. (eds.) (1997). *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning*. Los Alamitos, CA: IEEE Computer Society Press.

Morris, R., and Khatib, L. (eds.) (1999). Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning. Los Alamitos, CA: IEEE Computer Society Press.

Mortaz, R.; Renz, J., and Woter, D. (2000). Qualitative Spatial Reasoning About Line Segments. W.Horn (ed.): ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press, Amsterdam, 2000

Memmi, G., and Vautherin (1987). Analysing Nets by the Invariant Methos, *Lecture Notes in Computer Science,* 254, Springer, Berlin Heidelberg, New York, 300-336.

Nebel, B., and Burckert, H. J. (1995). Reasoning about temporal relations: a maximal tractable subset of Allen's interval algebra. *J. of the ACM*, 42(1):43-66.

Newton-Smith, W. H. (1980). The Structure of Time. London, U.K.: Routledge and Kegan.

Onder, N., and Pollack, M. E. (1999). Conditional, Probabilistic Planning: A Unifying Algorithm and Effective Search Control Mechanisms, in *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99).*

Peterson, J. L. (1981). *Petri Net Theory and the Modelling of Systems*. Prentice-Hall.

Pnueli, A. (1977). The temporal logic of programs, in *Proc. 18th IEEE Symp. Foundations Computer Science*, 1977, pp. 46–67.

Pollack, M. E., and Horty, J. F. (1999). There's More to Life than Making Plans: Plan Management in Dynamic, Multi-Agent Environments, *AI Magazine*, 20(4):71-84.

Prior, A. N. (1967). *Past, Present and Future*. Oxford, U.K.: Clarendon.

Prior, A. N. (1955). Diodoran modalities," Phil. Quart., vol. 5, pp. 205–213.

Quine, W. V. (1965). *Elementary Logic*, revised ed. New York: Harper and Row.

Renz, J., and Nebel, B. (1998) Efficient methods for qualitative spatial reasoning, in *Proceedings ECAI-98*, pp. 562–566, Brighton.

Randell, D.; Cui, Z., and Cohn, A. (1992). A spatial logic based on regions and connection. In Proceedings KR-92, pages 165--176, San Mateo. Morgan Kaufmann.

Rauf, I., and Zaidi, A. K. (2002). A Temporal Programmer for Revising Temporal Models of Discrete-Event Systems, in: *Proc. of 2002 IEEE International Conference on Systems, Man, and Cybernetics*, Hemmamat, Tunisia.

Reichgelt, H. (1989). A Comparison of First Order and Modal Logics of Time. In P. Jackson, H. Reichgelt & F. van Harmelen (eds.), *Logic Based Knowledge Representation*. Cambridge, MA: MIT Press, 143–176.

Reisig, W. (1991). Petri Nets and Algebraic Specifications. Theor. Comput. Sci. 80(1): 1-34 .

Renz, J., and Nebel, B. (1998). Efficient methods for qualitative spatial reasoning. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, Brighton, UK, 562-566,Wiley.

Rescher, N., and Urquhart, A. (1971). *Temporal Logic*. Berlin, Germany: Springer-Verlag.

Rosenschein, S. J., and Kaelbling, L. P. (1996). A situated view of representation and control. In P. E. Agre and S. J. Rosenschein, editors, *Computational Theories of Interaction and Agency*, 515-540. The MIT Press: Cambridge, MA.

Russell, S., and Norvig, P. (1995). *Artificial Intelligence A Modern Approach*, Prentice Hall, Upper Saddle River, NJ.

Haider, S.; Zaidi, A. K., and Levis, A. H. (2005). On Temporal Analysis of Timed Influence Nets using Point Graphs. To be presented in 18th International FLAIRS Conference, Florida, May 2005.

Shoham, Y. (1987). Temporal Logics in Artificial Intelligence: Semantical and Ontological Considerations. *Artificial Intelligence* 33: 89–104.

Stock, O. (ed) (1997). *Spatial and Temporal Reasoning*, Kluwer Academic Publishers.

Schwalb, E. (1998). Temporal Reasoning With Constraints. *A Ph.D. thesis, UCI, ICS.*

Schwalb, E., and Dechter, R. (1997). Processing disjunctions in temporal constraint networks. *Artificial Intelligence, 93(1/2):29-61.*

Schwalb, E., and Vila, L. (1998). Temporal constraints: A survey. *Constraints, 3,129--149.*

Tsamardinos, I.; Pollack, M. E., and Horty, J. F. (2000). "Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches," *Proceedings of the 5th International Conference on AI Planning Systems*, Breckenridge, CO.

Van Beek, P. (1990). Exact and Approximate Reasoning About Qualitative Temporal Relations. *Technical Report TR 90-29*, Department of Computing Science, University of Alberta.

Van Beek, P. (1992). Reasoning about qualitative temporal information. *Artificial Intelligence, 58:297—326.*

Van Benthem, J. F. A. K. (1991). *The Logic of Time (second edition)*. Reidel.

Vila, Ll. (1994). An Instant-Period Based Theory of Time. In R. Rodriguez (ed.), *Proceedings of the Workshop on Spatial and Temporal Reasoning in ECAI 94.*

Vilain, M. (1982). A system for reasoning about time. In *Proceedings of AAAI-82*, Pittsburgh, PA, 197-201.

Vilain, M.; Kautz, H., and van Beek, P. (1990). Constraint propagation algorithms for temporal reasoning: a revised report in *Readings in Qualitative Reasoning about Physical Systems*, San Mateo, CA, Morgan Kaufman, 373-381.

Warshall, S. (1962). A theorem on boolean matrices, *Journal of the ACM*, 9, 1, (1962), 11-12.

Williams, B.C., and Nayak, P.P. (1997). A Reactive Planner for Model-based Execution. In *Proceedings of the Fifteen International Joint Conference on Artificial Intelligence (IJCAI--97).*

Yao, Y. "A Petri net model for temporal knowledge representation and reasoning," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 1374–1382, Sept. 1994.

Zaidi, A. K. (1999). On Temporal Logic Programming Using Petri Nets, *IEEE Transactions on SMC*, Part A, 28, 3.

Zaidi, A. K., and Levis, A. H. (2001). TEMPER: A Temporal Programmer for Time-sensitive Control of Discrete-event Systems, *IEEE Transaction on Systems, Man, and Cybernetics,* 31, 6, 485-496.

Zaidi, A. K.; Rizvi, K. H., and Hussain, S. (2003). On Spatial Modeling of Discrete Event Systems Using Point-Interval Logic, in: *Proc. of IEEE SMC 2003*, Washington DC, 2003.

Zaidi, A. K. (2003). Qualitative and Quantitative Spatiotemporal Knowledge Representation and Reasoning Using Point Graphs. *Proc. of Workshop on Spatial and Temporal Reasoning*, 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico.

Zaidi, A. K., and Wagenhals, L. W. 2004. Planning Temporal Events Using Point Interval Logic. Special Issue of *Mathematical and Computer Modeling*. Forthcoming.

Zaidi, A. K., and Ishaque, M. (2005).